VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro správu a synchronizaci účtů na unixových systémech

Account Administration and Synchronization on Unix Systems

2012 Václav Bortlík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Václav Bortlík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroj pro správu a synchronizaci účtů na unixových systémech
Account Administration and Synchronization on Unix Systems

Zásady pro vypracování:

Cílem této práce je návrh a implementace nástroje pro správu a synchronizaci účtů na unixových systémech. Navržená implementace by měla přinést centrální správu účtů, skupin a přístupových práv. Účty by měly být distribuovány na podřízené systémy. Navržený systém nesmí vyžadovat on-line spojení podřízených systémů s nadřízeným centrem.

1. Nastudování a porovnání alternativních technologií pro vzdálenou správu, jejich výhody a nevýhody.
2. Návrh a implementace nástroje pro správu a synchronizaci uživatelských účtů a skupin. Řešení bude realizováno pro nejpoužívanější komerční a nekomerční unixové systémy.
3. Návrh a vytvoření přehledného uživatelského rozhraní pro jednoduchou správu.
4. Nasazení správy účtů na několik systémů, testování spolehivosti a bezpečnosti.
5. Postup nasazení systému, zkušenosti.
6. Uživatelská a systémová dokumentace, návrhy školení.

Seznam doporučené odborné literatury:

Ray Deborah S.: Unix podrobný průvodce, Grada, ISBN 978-80-247-2125-5
Manuálové stránky jednotlivých implementací Unixů

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2012 ..... Václav Rolf

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 ..... Václav Rolf

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by nevznikla. Poděkování patří především vedoucímu práce Ing. Petru Olivkovi za vstřícný přístup a mnoho cenných rad. Dále bych rád poděkoval svým spolupracovníkům za odborné konzultace a nápady během tvorby aplikace. Nerad bych jmenovitě uváděl všechny - nechtěl bych totiž někoho vynechat.

Abstrakt

Cílem této práce je vytvoření nástroje pro jednotnou a centralizovanou správu uživatelských účtů na UNIXových systémech v produkčním prostředí mezinárodní společnosti Tieto. Ve stávajícím prostředí není doposud žádný nástroj, který by centralizovanou správu umožňoval. Důležité je, aby podřízené systémy nevyžadovaly přímé spojení k centrálnímu prvku během ověřování při přihlašování uživateli na podřízené systémy. Nástroj bude postupně testován a poté integrován v produkčním prostředí společnosti.

Klíčová slova: centrální správa uživatelských účtů, správa identity, správa rolí, IDM, UNIX, Linux, Solaris, AIX, HP-UX, Perl, Expect, paralelní programování

Abstract

The main idea of this Thesis is the creation of a tool for the unified and centralized account management on UNIX systems in production environment of the international company Tieto. At the moment there is no tool, which allows centralized account management. It's important, that slave systems must not require direct connection to central management system during user login authentication process to slave systems. The tool will be tested and then will be integrated into company production environment.

Keywords: centralized account management, identity management, role management, IDM, UNIX, Linux, Solaris, AIX, HP-UX, Perl, Expect, Parallel programming model

Seznam použitých zkratek a symbolů

AS	–	Authentication server
TGT	–	Ticket Granting Ticket
RSA	–	authors of an algorithm - Rivest, Shamir and Adleman
NIS	–	Network Information Service
LDAP	–	Lightweight Directory Access Protocol
PAM	–	Pluggable authentication module
SSO	–	Single sign-on
CPAN	–	Comprehensive Perl Archive Network
MVC	–	Model-view-controller
ER	–	Entity-relationship
FC	–	Fibre Channel

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Téma správy uživatelských účtů v podnikové počítačové síti jednou začne řešit každý její správce. Jeho snahou by mělo být nasazení systému pro snadnou správu uživatelských účtů. Jedním z nejdůležitějších základních požadavků je centralizace celého systému do jednoho bodu. Dalším nárokem je automatizace procesů mezi uživateli, uživatelskými rolemi a podřízenými systémy. Spojení těchto dvou požadavků vznikne systém s jednoduchou správou a snadnou distribucí příkazů na podřízené systémy. Sníží se časová náročnost a zároveň zvýší produktivita práce správce systémů.

Účelem realizace této práce je implementace systému správy uživatelských účtů pro komerční a nekomerční Unixové systémy. V nástroji budou implementovány jednoduché funkce pro vytváření a odebírání účtů (uživatelů a skupin), změna hesel uživatelů. Vše bude realizováno za předpokladu nezávislosti podřízených systémů s nadřízeným centrálním systémem.

Práce je rozdělena na několik kapitol. První z nich je nastudování a porovnání alternativních technologií správy uživatelských účtů, popis jejich vlastností, výhody a nevýhody. Následující část se zabývá fází návrhu a implementace systému, včetně jednoduchého uživatelského rozhraní. Dále následuje postup nasazení systému. Součástí práce je uživatelská a systémová dokumentace.

Ve firmě spravujeme dohromady několik tisíc serverů a vytvořit účet pro každého zaměstnance na těchto serverech vyžaduje obrovskou časovou náročnost. Samozřejmě všichni zaměstnanci nemají přístup na všechny servery, ale i tak postrádáme nástroj, kterým bychom mohli spravovat účty z jednoho místa. Od systému si slibuji zvýšení efektivity a produktivity práce, místo toho, abychom se zabývali přidáváním a odebíráním účtů.

2 Alternativní technologie správy uživatelských účtů

Ve středních a větších firmách jejich zaměstnanci používají každý den několik desítek aplikací, ke kterým se přihlašují. Všem těmto uživatelům je třeba zpřístupnit zdroje, které poté využívají během práce. Existují systémy zabývající se správou uživatelů a zdrojů, které pak přiřazují do příslušných rolí. U všech těchto systémů se setkáváme s pojmem identita (jednoznačné určení jedinečného objektu). Je potřeba najít vhodné řešení pro správu identit, které poskytne balík procesů a technologií pro řízení bezpečného přístupu k informacím a informačním zdrojům ve firmě.

Mezi základní vlastnosti správy identit patří:

- centralizovaná správa
- vytváření a přiřazování uživatelů a zdrojů (systémy a aplikace) do rolí
- možnost provádění aktualizace a synchronizace hesel uživatelů
- možnost výpisu a evidence aktuálního stavu jednotlivých objektů

Nejznámější komerční systémy správy identit:

- CA Identity Manager
- IBM Tivoli Identity Manager
- Sun Identity Manager / Oracle Identity Management
- Imprivata OneSign
- Novell Identity Manager

Vedle správy identit je také nutno zajistit správu přístupu uživatelů vůči koncovým systémům. Většina systémů zabývajícím se tímto problémem, obsahuje funkce pro správu identit a zároveň řeší správu přístupu. Oba tyto systémy jsou navzájem nepostradatelné. Správa přístupu může být řešena několika autentizačními mechanismy.[1]

2.1 Lokální databáze

Standardně se uživatelé v unixovém prostředí ověřují proti lokální textové databázi nacházející se v `/etc/passwd`. Zde se nachází základní informace o uživateli a systémové proměnné uživatele. Některé systémy, zejména starší HP-UX systémy, jej také používají pro uchovávání otisku hesla uživatele. Nové moderní systémy mají z bezpečnostního hlediska uloženy otisk hesla v souboru `/etc/shadow`, který je přístupný pouze uživateli `root`. Uživatelské skupiny jsou uloženy v souboru `/etc/group`.[2]

[1] Informace v této části kapitoly pocházejí z tutoriálu Petra Hanáčka a Jana Staudka, Správa identity[2].
[2] Informace v této podkapitole pocházejí z Linuxových manuálových stránek[10].

2.2 NIS

NIS je zkratkou pro protokol síťové informační služby. Jeho cílem je poskytovat informace, které mají být známe v celé počítačové síti všem strojům v síti. NIS se tará například o distribuci údajů v již zmíněných souborech `/etc/passwd` (přihlašovací jména, domovské adresáře, hesla) a /etc/group (informace o skupinách). Pokud je heslo zaznamenáno v NIS databázi, bude možno se přihlásit pod jedním heslem na všech strojích, na kterých běží NIS klient.[3]

2.3 Adresáře a adresářové služby

Adresář slouží k organizování a sdružování dat do skupin, a to z důvodu, aby se v nich mohl uživatel snáze orientovat. V podstatě jsou to jakési kontejnery pro ukládání dat, či specializované databáze, ve kterých lze uchovat a především vyhledávat velké množství informací různého charakteru. Na rozdíl od relačních databází jsou adresáře určeny především pro vyhledávání, nikoliv pro klasické transakce (časté ukládání a změny dat, případně velmi komplikované dotazy). V normálním životě se lze setkat s offline adresáři, které jsou nejčastěji k dispozici v tištěné verzi (telefonní seznamy, apod.). Při práci s počítači se používají online adresáře.[4]

2.3.1 LDAP

Nejčastější podobou autentizace z centrálního úložiště je adresářová struktura LDAP. Zde autentizace uživatelů probíhá ve spojení s adresářovým serverem, na kterém probíhají operace ukládání a přístupu k datům. Jednotlivé položky jsou na serveru ukládány formou záznamů do stromové architektury. LDAP je vhodný pro práci s informacemi o uživatelích. Provádění požadavku autentizace probíhá na bázi server-klient dotazování. Otisk hesla je uložen na straně serveru a proto je nutné síťové připojení. Pro zprovoznění LDAP autentizace na UNIX systémech je nutná instalace knihoven a nastavení konfiguračních souborů `/etc/nsswitch.conf`. Například v PAM je třeba nastavit adresu konkrétního LDAP serveru.[5]

2.4 Single sign-on

Single sign-on systémy (SSO) zvyšují komfort rychlého a bezpečného přístupu do několika relací nezávislých IT systémů. Základní princip je v tom, že uživatel si pamatuje jen jednu sadu přihlašovacích údajů – jedno přístupové jméno a k tomu příslušné heslo. Veškeré operace, kromě prvotní ověření identity probíhají bez zásahu uživatele, který tak není zdržován ve své činnosti. Použitím SSO systémů zamezíme tomu, aby uživatel si neukládal hesla do jiných snadno zneužitelných míst (papír a tužka, textový soubor,

[3] Informace v této podkapitole jsou převzaty z knihy Hala Sterna, Mika Eislera a Ricarda Labiagy, Managing NFS and NIS, 2nd Edition[3].

[4] Tato podkapitola je převzata z diplomové práce Karla Beňáka[1].

[5] Tato podkapitola je převzata z knihy Geralda Cartera, LDAP System Administration[4].

apod.). Pro zvýšení bezpečnosti se SSO systém používá ve spojitosti s dalším autentizačním prvkem – např. automaticky generovaný PIN v určitém časovém intervalu (RSA SecurID token).[6]

Pro přístup k jednotlivým objektům v systému, se používají silná automaticky generovaná hesla, které si uživatel nemusí pamatovat. Další výhodou SSO systémů je možnost změny zapomenutého hesla a pravidelných změn před jeho expirací.

2.5 Kerberos

Kerberos je síťový autentizační protokol, umožňující uživatelům v nezabezpečené počítačové síti prokázat svou identitu. Stejně jako v případě LDAP protokolu se jedná o klient-server autentizaci, kde si oba subjekty vzájemně prokazují svou identitu. Základním prvkem Kerbera je principal, který představuje identitu daného objektu. Jím může být uživatel, služba, nebo také počítač.

Způsob autentizace je založen na principu tiketů rozdávaných centrálním autentizačním serverem (AS). Všechny protokoly Kerbera využívají symetrické kryptografie. Veškeré zprávy jsou šifrovány i dešifrovány jediným sdíleným klíčem mezi uživatelem a autentizačním serverem. V systému Kerberos je možnost přihlašování pomocí SSO. Uživatel se jen jednou prokáže vůči autentizačnímu serveru a požádá o takzvaný základní tiket, Ticket Granting Ticket (TGT), který následně využívá k dalším službám systému během prokazování identity.

Kerberos je možno využít v LDAP serveru. Jejich cílem je možnost autentizace LDAP uživatelů přes Kerberos s využitím systému jednotného hesla (SSO). LDAP nemusí znát žádná hesla uživatelů, protože jejich autentizaci obstará samotný Kerberos. Stejně jako u služby LDAP, je u Kerbera nutná konfigurace. Tu musíme provést jak na straně serveru (vytvoření identit, klíčů pro ověření uživatelů), tak musí být rovněž podpora ze strany klientského systému (konfigurace PAM modulu).[7]

2.6 Výběr systému

Nevýhodou řízení přístupu pomocí centralizovaného úložiště je podmínka dostupnosti síťového přípojení mezi klientem a autentizačním serverem. V případě výpadku sítě se nebude možno připojit ke koncovému systému. Klient musí podporovat příslušný protokol a musí být zkonfigurován, což se někdy neobejde bez problémů. Jediná metoda řízení přístupu, která je vhodná pro vzdálenou správu platformě nezávislých systémů, je pomocí lokálních databází. Komerční systémy správy identit využívají knihovny pro správu uživatelů pomocí lokálních databází. Pro jeho nasazení je nutné zakoupení licence a vyškolení několika zaměstnanců, kteří se budou o systém starat. Procesy mezi výběrem a nasazení takového systému ve velké organizaci může trvat i několik let. Rozhodl jsem se navrhnout vlastní řešení systému správy identit, které bude dostatečně splňovat podmínky a může být během několika měsíců nasazeno do ostrého provozu.

[6] Tato podkapitola je převzata z tutoriálu Petra Hanáčka a Jana Staudka, Správa identity[2].

[7] Tato podkapitola je převzata z knihy Jasona Garmana, Kerberos: The Definitive Guide[5].

3 Návrh a analýza

Veškeré podklady pro vytvoření návrhu, analýzy, tvorby diagramů a implementace datového modelu, jsem čerpal ze studijního materiálu Jany Šarmanové[8].

3.1 Podrobná specifikace

V současné době se uživatelské účty ve firmě vytvářejí manuálně, případně se používají skripty spouštěné lokálně na systému, kde se účty vytvářejí. V této kapitole jsou rozebrány veškeré požadavky na tvorbu aplikace - její chování, kdo bude se systémem pracovat, funkce systému, apod.

3.1.1 Požadavky na vnitřní architekturu systému

Systém se skládá ze třech základních částí, a to účty (accounts), zdroje (resources) a role (roles). Účty tvoří uživatelé (users) a oddělení (departments) ve firmě. Uživatelé budou spadat do oddělení, ve kterém se zrovna nacházejí. Oddělení jsou rozdělena podle vedoucího daného úseku. Zdroje se skládají z podřízených systémů (systems) a zákazníků (customers). Vztah mezi podřízeným systémem a zákazníkem je v podobné relaci jako v předchozím případě. To znamená, že každý systém bude patřit právě jednomu zákazníkovi a bude unikátní. Poslední důležitou částí systému jsou role, tvořené z uživatelských rolí (user roles) a systémových rolí (system roles). Již podle názvu můžeme usoudit, že uživatelské role budou ve vztahu s uživateli a systémové role ve vztahu k podřízeným systémům. Podrobný popis všech entit je součástí datové analýzy v kapitole 3.2.

Celý systém bude běžet na dedikovaném linuxovém stroji, který budeme nazývat řídicí (management) server. Hostovaný operační systém řídicího serveru bude Red Hat Enterprise Linux. Tento server bude výchozím bodem pro přístup na všechny podřízené systémy v produkční podnikové síti. Komunikace bude probíhat tak, že se z řídicího serveru vytvoří spojení výchozím komunikačním protokolem na podřízené systémy. Po navázaní spojení se postupně vykoná sled nachystaných příkazů. Vzhledem k tomu, že podřízené systémy jsou různých unixových platforem (SunOS, AIX, Linux, HP-UX), je potřeba dodržet jejich platformní nezávislost během spouštění příkazů na těchto systémech. Pro každou platformu se budou vykonávat příkazy s odlišnými parametry se stejnými, nebo podobnými výsledky. V případě vykonávání úloh, které budou vyžadovat spojení na více podřízených systémů, budou spojení navázána současně, aby byla doba provádění úloh co nejkratší.

U každého vykonaného příkazu se bude zachytávat jeho výstup (standardní i chybový), který se poté pošle zpět řídicímu serveru ke zpracování.

3.1.2 Kdo bude se systémem pracovat

Se systémem budou pracovat vyškolení správci (administrator) společnosti, kteří budou mít přístup na řídicí server. Tito správci budou jediným provozovatelem systému

[8] Jana Šarmanová, Databázové a informační systémy[6].

a zároveň jeho jediným společným aktérem. Na řídicí server se správci dostanou pomocí zabezpečeného komunikačního protokolu SSH přes příkazovou řádku. Ovládání systému bude pomocí příkazů v textové konzoli se zadanými parametry.

3.1.3 Vstupy do systému

- Vytvoření nového zákazníka
- Vytvoření nového systému
- Vytvoření nového oddělení
- Vytvoření nového uživatele
- Vytvoření nové systémové role
- Vytvoření nové uživatelské role
- Přiřazení uživatele do uživatelské role
- Přiřazení systému do systémové role
- Vytvoření nové úlohy
- Smazání zákazníka
- Smazání systému
- Smazání oddělení
- Smazání uživatele
- Smazání systémové role
- Smazání uživatelské role
- Odebrání uživatele z uživatelské role
- Odebrání systému ze systémové role
- Smazání úlohy
- Změna hesla uživatele

3.1.4 Výstupy ze systému

- Výpis seznamu všech zákazníků
- Výpis seznamu všech systémů
- Výpis seznamu všech oddělení
- Výpis seznamu všech uživatelů
- Výpis seznamu systémových rolí
- Výpis seznamu uživatelských rolí
- Výpis seznamu relací uživatelů a uživatelských rolí
- Výpis seznamu relací systémů a systémových rolí
- Výpis seznamu všech nedokončených úloh

3.1.5 Funkce systému

Funkcemi systému jsou všechny vstupy a výstupu. Nad všemi těmito tabulkami je možnost provádění třídění a vyhledávání.

3.1.6 Vazby na okolí systému

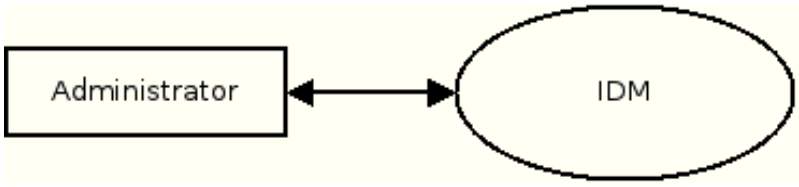
Data do systému budou zadávána prostřednictvím provozovatele systému.

3.1.7 Nefunkční požadavky

Při řešení je požadováno dodržení programovacích standardů. Zdrojový kód bude opatřen dostatečným počtem komentářů pro jeho přehledné čtení. Pro implementaci bude využíván skriptovací programovací jazyk Perl a SŘBD SQLite. Systém bude v některých částech zdrojového kódu pracovat s velmi citlivými daty (přenášení hesel uživatelů), a proto se bude muset dodržet vysoká bezpečnost a důslednost při jeho implementaci.

3.1.8 Kontextový diagram

Na obrázku 1 máme znázorněný kontextový diagram, který nám popisuje vnější chování systému.



Obrázek 1: Kontextový diagram

3.2 Datová analýza

3.2.1 Podrobná specifikace vztahů mezi entitami

V kapitole podrobné specifikace 3.1 jsme se dočetli, že se systém bude skládat ze tří základních částí, a to účtů (accounts), zdrojů (resources) a rolí (roles). Ty budou dále rozděleny do následujících entit. Účty tvoří uživatelé (users) a oddělení (departments). Při vytváření uživatele vybereme právě jedno oddělení, ve kterém se uživatel nachází. Každé oddělení bude moci evidovat několik uživatelů. Ale je možnost, že se v daném oddělení nebude zrovna nacházet žádný uživatel. Podobný vztah je rovněž mezi zákazníky (customers) a podřízenými systémy (systems). Zákazník může mít několik systémů, ale také žadný. Každý podřízený systém bude patřit právě jednomu zákazníkovi.

Mezi uživateli a uživatelskými rolemi (user roles) platí vazba M:N. Jakýkoliv živatel může být reprezentován v několika uživatelských rolích. Stejná relace platí mezi entitami podřízených systémů a systémových rolí (system roles). Čili systémy mohou být reprezentovány několika rolemi, ale také žádnou a zároveň každá systémová role bude moci obsahovat jakýkoliv počet podřízených systémů.

Jelikož relace tabulek uživatel - uživatelská role a systém – systémová role mají vazbu M:N, musíme zajistit, abychom určitého uživatele nepřidali na stejný podřízený systém několikrát ve stejné roli. To provedeme tak, že při vyváření systémové role vybereme určitou uživatelskou roli. Každá uživatelská role bude moci být součástí několika systémových rolí. Pokud budeme chtít přidat podřízený systém do systémové role, automaticky na tomto systému vytvoříme skupinu této role. Pokud zařadíme uživatele do uživatelské role, a tuto roli budeme mít v několika systémových rolích, uživatel bude na systémech přidán pro každou roli do jiné skupiny. Může nám také vzniknout konflikt, kde jednoho uživatele přiřadíme do více uživatelských rolí, a ty pak přiřadíme do systémových rolí se shodnými názvy skupin. Uživatel by poté byl přidán vícekrát na systémy se stejnou skupinou. Tuto skutečnost je nutno ošetřit v rámci implementace.

Poslední relací tabulek jsou úlohy (tasks). Ty jsou reprezentovány uživateli, podřízenými systémy a systémovými rolemi. Každá relace úlohy bude obsahovat právě jeden podřízený systém a maximálně jednu systémovou roli a jednoho uživatele. Uživatele nemusí obsahovat z toho důvodu, že budeme podřízené systémy přidávat do systémové role, které nejsou ve vazbě s žádnými uživateli. Systémovou roli nebude obsahovat například při změně hesla uživatelů. Každá relace uživatele, podřízeného systémy a systémové role, může být ve více úlohách současně.

3.2.2 Podrobná specifikace atributů entit

3.2.2.1 Department

Atribut	Popis
department_id	jednoznačná identifikace oddělení
department_name	jméno oddělení
private_key_dir	adresář s veřejnými klíči uživatelů
department_comment	komentář

3.2.2.2 User

Atribut	Popis
uid	jednoznačná identifikace uživatele
department_id	cizí klíč jednoznačné identifikace oddělení
login_name	přihlašovací jméno
first_name	jméno uživatele
last_name	příjmení uživatele
password_hash	heslo (resp. jeho otisk)
phone	číslo telefonu
email	adresa elektronické pošty
user_private_key	odkaz na soukromý klíč uživatele
user_comment	komentář

3.2.2.3 Customer

Atribut	Popis
customer_id	jednoznačná identifikace zákazníka
customer_name	jméno zákazníka
patrol_username	jméno uživatele patrol
private_login_flag	příznak vyhrazeného přihlašovacího jména
customer_comment	komentář

3.2.2.4 System

Atribut	Popis
system_id	jednoznačná identifikace systému
customer_id	cizí klíč jednoznačné identifikace zákazníka
fqdn	plně specifikované doménové jméno systému
platform	typ unixové platformy pracovního prostředí
dhome	cesta výchozího domovského adresáře uživatelů
shell	výchozí interpret příkazového řádku
protocol	protokol pro navázání komunikace
port	port protokolu
account_type	typ účtu pro přihlášení na podřízený systém
account_name	jméno účtu přihlášení podřízeného systému
database_version	verze databáze hesel
last_jumppoint	předchozí jumppoint systém
jumppoint_flag	příznak jumppoint systému
customer_comment	komentář

3.2.2.5 User role

Atribut	Popis
user_role_id	jednoznačná identifikace uživatelské role
user_role_name	jméno uživatelské role
user_role_comment	komentář

3.2.2.6 System role

Atribut	Popis
system_role_id	jednoznačná identifikace systémové role
user_role_id	cizí klíč jednoznačné identifikace uživatelské role
system_role_name	jméno systémové role
gid	identifikační číslo skupiny na podřízených systémech
group_name	jméno skupiny na podřízených systémech
passwd_comment	společná část komentáře uživatelů dané skupiny
system_role_comment	komentář

3.2.2.7 User role mapping

Atribut	Popis
uid	cizí klíč jednoznačné identifikace uživatele
user_role_id	cizí klíč jednoznačné identifikace uživatelské role

3.2.2.8 System role mapping

Atribut	Popis
`system_id`	cizí klíč jednoznačné identifikace systému
`system_role_id`	cizí klíč jednoznačné identifikace systémové role

3.2.2.9 Task

Atribut	Popis
`operation`	druh úlohy
`system_id`	cizí klíč jednoznačné identifikace systému
`uid`	cizí klíč jednoznačné identifikace uživatele
`system_role_id`	cizí klíč jednoznačné identifikace systémové role

3.2.3 Seznam typů entit včetně atributů

[9] Department (**department_id**, department_name, private_key_dir, department_comment)
User (**uid**, *department_id*, login_name, first_name, last_name, password_hash, phone, email, user_private_key, user_comment)

Customer (**customer_id**, customer_name, patrol_username, private_personal_account_flag, customer_comment)
System (**system_id**, *customer_id*, fqdn, platform, dhome, shell, protocol, port, account_type, account_name, database_version, last_jumppoint, jumppoint_flag, system_comment)

User_role (**user_role_id**, user_role_name, user_role_comment)
System_role (**system_role_id**, *user_role_id*, system_role_name, gid, group_name, passwd_comment, system_role_comment)

User_role_mapping (***uid***, ***user_role_id***)
System_role_mapping (***system_id***, ***system_role_id***)

Task (**operation**, ***system_id***, ***uid***, ***system_role_id***)

3.2.4 Seznam typů vztahů

USER_IS_A_PART_OF_DEPARTMENT (User, Department)
SYSTEM_IS_A_PART_OF_CUSTOMER (System, Customer)

USER_IS_ASSIGNED_TO_USER_ROLE_MAPPING (User, User_role_mapping)
USER_ROLE_IS_ASSIGNED_TO_USER_ROLE_MAPPING (User_role, User_role_mapping)

[9] Tučně zvýrazněné jsou primární klíče, kurzívou cizí klíče

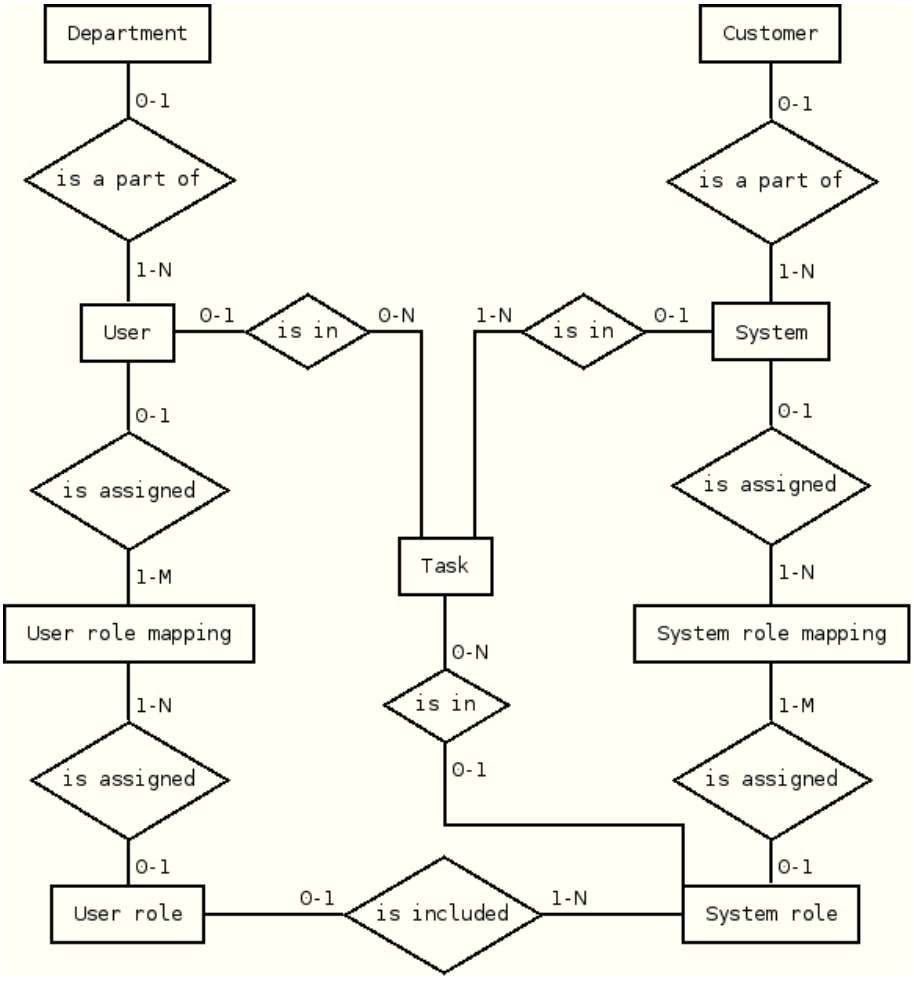
SYSTEM_IS_ASSIGNED_TO_SYSTEM_ROLE_MAPPING (System, System_role_mapping)
SYSTEM_ROLE_IS_ASSIGNED_TO_SYSTEM_ROLE_MAPPING (System_role, System_role_mapping)

USER_ROLE_IS_INCLUDED_IN_SYSTEM_ROLE (User_role, System_role)

TASK_IS_IN_SYSTEM(Task, System)
TASK_IS_IN_USER(Task, User)
TASK_IS_IN_SYSTEM_ROLE(Task, System_role)

3.2.5 ER diagram

Na obrázku 2 máme znázorněný ER diagram, který nám popisuje abstraktní a konceptuální znázornění dat systému.



Obrázek 2: ER diagram

3.2.6 Datový slovník

3.2.6.1 Department

Jméno	typ	délka	klíč	null	index	IO
department_id	Integer	255	yes	no	yes	
department_name	Varchar	255	no	no	yes	UNIQUE
private_key_dir	Varchar	255	no	no	no	
department_comment	Varchar	5000	no	yes	no	

3.2.6.2 User

Jméno	typ	délka	klíč	null	index	IO
uid	Integer	255	yes	no	yes	
department_id	Integer	255	yes	no	yes	foreign key
login_name	Varchar	255	no	no	yes	UNIQUE
first_name	Varchar	255	no	no	no	
last_name	Varchar	255	no	no	no	
password_hash	Varchar	255	no	yes	no	
phone	Varchar	255	no	yes	no	
email	Varchar	255	no	yes	no	
user_private_key	Varchar	255	no	yes	no	
user_comment	Varchar	5000	no	yes	no	

3.2.6.3 Customer

Jméno	typ	délka	klíč	null	index	IO
customer_id	Integer	255	yes	no	yes	
customer_name	Varchar	255	no	no	yes	UNIQUE
patrol_username	Varchar	255	no	yes	no	
private_login_flag	Integer	255	no	no	no	
customer_comment	Varchar	5000	no	yes	no	

3.2.6.4 System

Jméno	typ	délka	klíč	null	index	IO
system_id	Integer	255	yes	no	yes	
customer_id	Integer	255	yes	no	yes	foreign key
fqdn	Varchar	255	no	no	yes	UNIQUE
platform	Varchar	255	no	no	no	
dhome	Varchar	255	no	no	no	
shell	Varchar	255	no	no	no	
protocol	Integer	10	no	no	no	
port	Integer	10	no	no	no	
account_type	Integer	10	no	no	no	
account_name	Varchar	255	no	yes	no	
database_version	Integer	10	no	yes	no	
last_jumppoint	Integer	255	no	yes	no	system id
jumppoint_flag	Integer	10	no	no	no	
customer_comment	Varchar	5000	no	yes	no	

3.2.6.5 User role

Jméno	typ	délka	klíč	null	index	IO
user_role_id	Integer	255	yes	no	yes	
user_role_name	Varchar	255	no	no	yes	UNIQUE
user_role_comment	Varchar	5000	no	yes	no	

3.2.6.6 System role

Jméno	typ	délka	klíč	null	index	IO
system_role_id	Integer	255	yes	no	yes	
user_role_id	Integer	255	yes	no	yes	foreign key
system_role_name	Varchar	255	no	no	yes	UNIQUE
gid	Integer	5	no	no	no	
group_name	Varchar	255	no	no	no	
passwd_comment	Varchar	255	no	no	no	
system_role_comment	Varchar	5000	no	yes	no	

3.2.6.7 User role mapping

Jméno	typ	délka	klíč	null	index	IO
uid	Integer	255	yes	no	yes	foreign key
user_role_id	Integer	255	yes	no	yes	foreign key

3.2.6.8 System role mapping

Jméno	typ	délka	klíč	null	index	IO
system_id	Integer	255	yes	no	yes	foreign key
system_role_id	Integer	255	yes	no	yes	foreign key

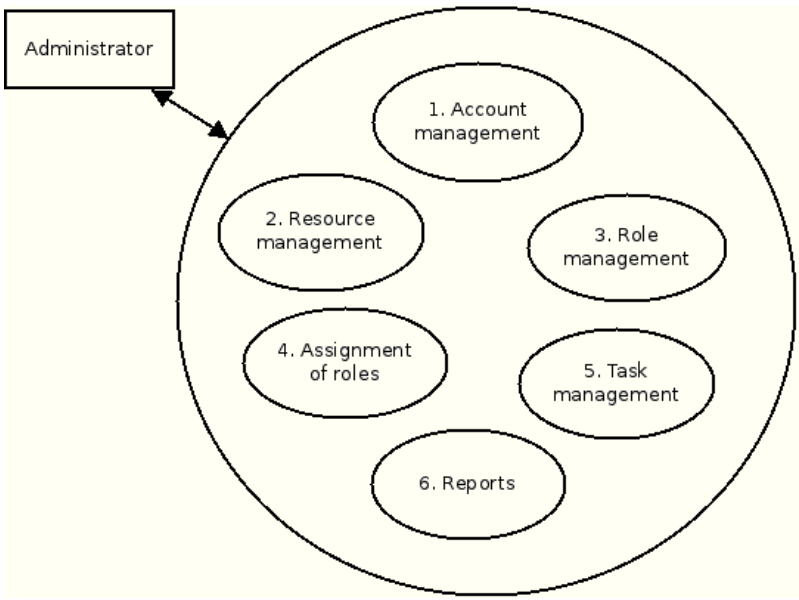
3.2.6.9 Task

Jméno	typ	délka	klíč	null	index	IO
operation	Varchar	255	yes	no	yes	
system_id	Integer	255	yes	no	yes	foreign key
uid	Integer	255	yes	no	yes	foreign key
system_role_id	Integer	255	yes	no	yes	foreign key

3.3 Funkční analýza

Funkční analýza je složena z diagramů datových toků a popisu jejich elementárních funkcí tvořených minispecifikacemi. V případě, že jakákoliv z akcí skončí neúspěchem, oznámíme ji uživateli a vrátíme se zpět.

Na obrázku 3 máme znázorněný DFD diagram úrovně 0, který dostaneme rozkladem kontextového diagramu z obrázku 1. DFD diagram úrovně 0 obsahuje základní funkce systému a jejich vztahy pomocí datových toků a pamětí.



Obrázek 3: DFD diagram úrovně 0

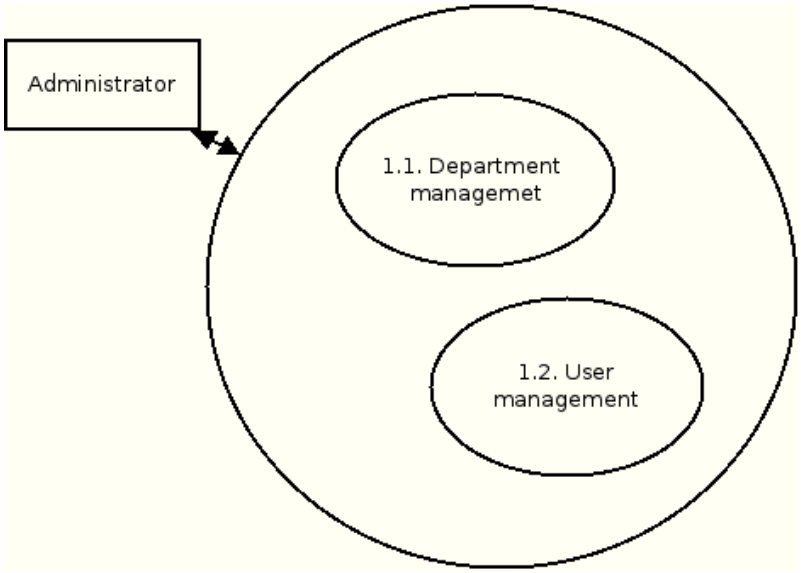
3.3.1 Správa účtů

Na obrázku 4 máme znázorněný DFD diagram úrovně 1 - správa účtů.

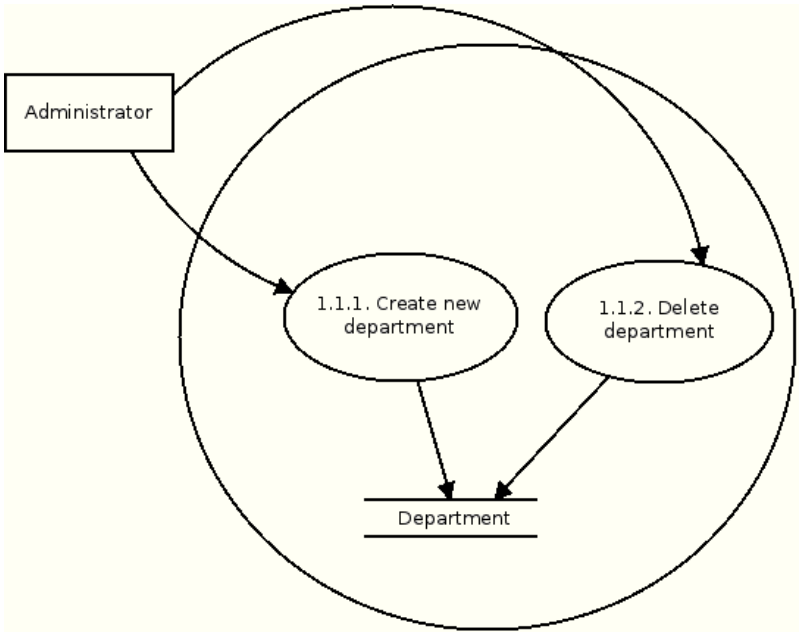
3.3.1.1 Správa oddělení Na obrázku 5 máme znázorněný DFD diagram úrovně 2 - správa oddělení.

3.3.1.1.1 Vytvoření nového oddělení

- generuj formulář `New_department`
- aktér zadá hodnoty všech povinných atributů tabulky `Department`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `department_id`



Obrázek 4: DFD diagram úrovně 1 - správa účtů



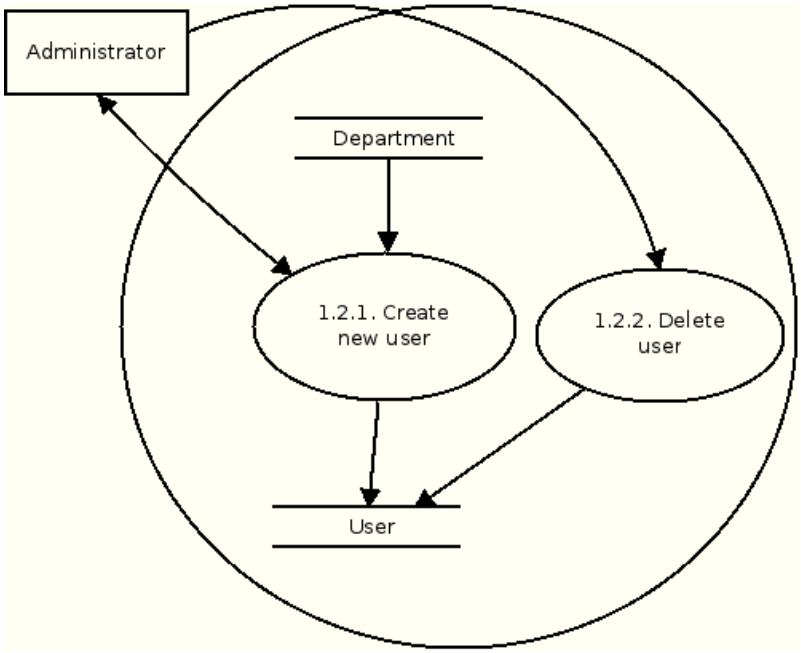
Obrázek 5: DFD diagram úrovně 2 - správa oddělení

- po úspěšném ověření a potvrzení aktérem se v tabulce `Department` vloží nová věta s primárním klíčem `department_id`

3.3.1.1.2 Smazání oddělení

- ověř existenci `department_id`
- ověř, že záznam klíče `department_id` není použit v ostatních tabulkách
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.1.2 Správa uživatelů Na obrázku 6 máme znázorněný DFD diagram úrovně 2 - správa uživatelů.



Obrázek 6: DFD diagram úrovně 2 - správa uživatelů

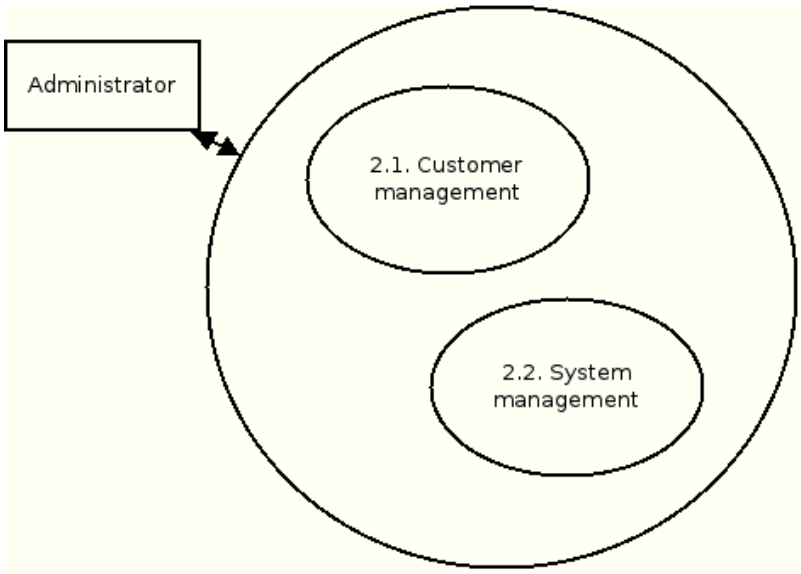
3.3.1.2.1 Vytvoření nového uživatele

- generuj formulář `New_user`
- aktér vybere jednu hodnotu atributu `department_id` a zadá hodnoty všech povinných atributů tabulky `User`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `uid`
- po úspěšném ověření a potvrzení aktérem se v tabulce `User` vloží nová věta s primárním klíčem `uid`

3.3.1.2.2 Smazání uživatele

- ověř existenci `uid`
- ověř, že záznam klíče `uid` není použit v ostatních tabulkách
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.2 Správa zdrojů

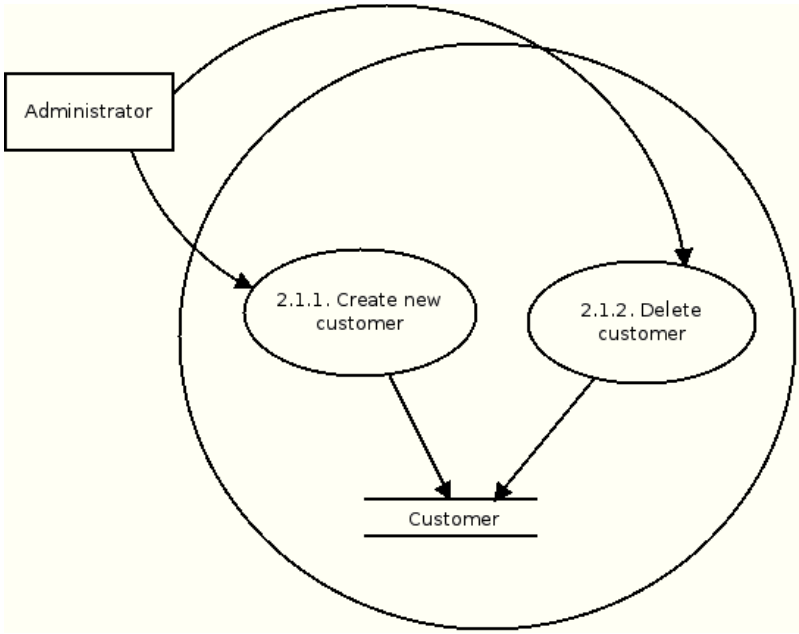


Obrázek 7: DFD diagram úrovně 1 - správa zdrojů

3.3.2.1 Správa zákazníků

3.3.2.1.1 Vytvoření nového zákazníka

- generuj formulář `New_customer`
- aktér zadá hodnoty všech povinných atributů tabulky `Customer`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `customer_id`
- po úspěšném ověření a potvrzení aktérem se v tabulce `Customer` vloží nová věta s primárním klíčem `customer_id`



Obrázek 8: DFD diagram úrovně 2 - správa zákazníků

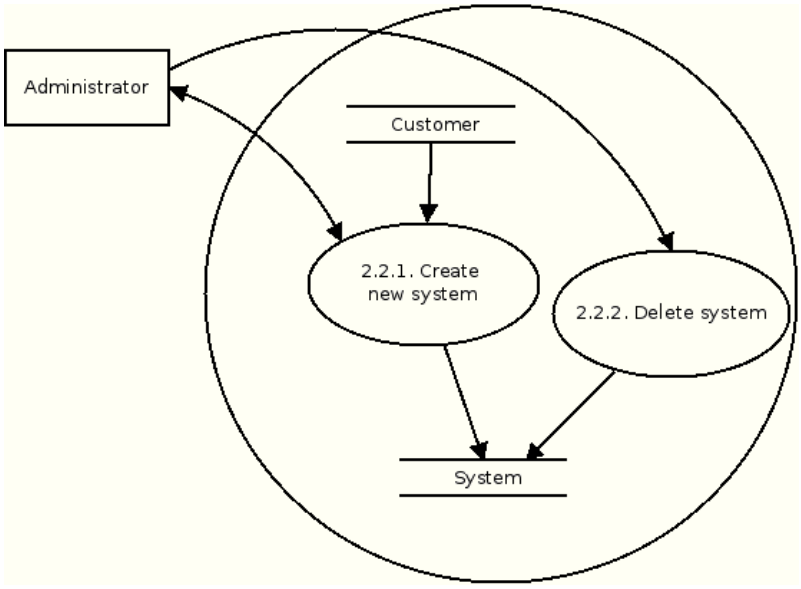
3.3.2.1.2 Smazání zákazníka

- ověř existenci `customer_id`
- ověř, že záznam klíče `customer_id` není použit v ostatních tabulkách
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.2.2 Správa podřízených systémů

3.3.2.2.1 Vytvoření nového podřízeného systému

- generuj formulář `New_user`
- aktér vybere jednu hodnotu atributu `customer_id` a zadá hodnoty všech povinných atributů tabulky `System`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `system_id`
- po úspěšném ověření a potvrzení aktérem se v tabulce `System` vloží nová věta s primárním klíčem `system_id`



Obrázek 9: DFD diagram úrovně 2 - správa systémů

3.3.2.2.2 Smazání podřízeného systému

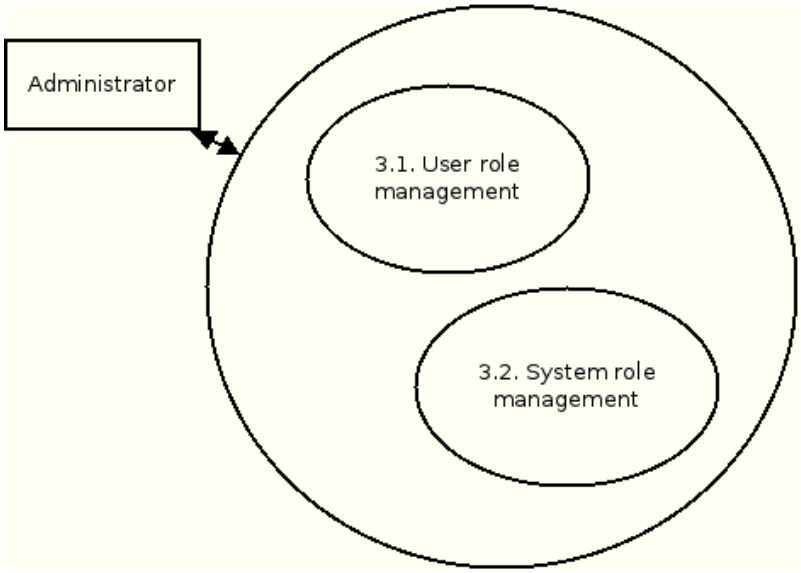
- ověř existenci `system_id`
- ověř, že záznam klíče `system_id` není použit v ostatních tabulkách
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.3 Správa rolí

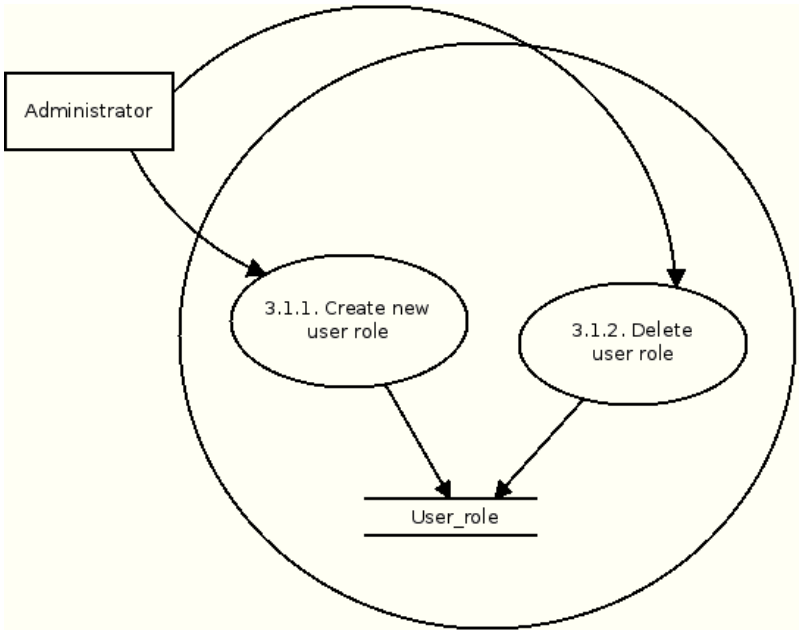
3.3.3.1 Správa uživatelských rolí

3.3.3.1.1 Vytvoření nové uživatelské role

- generuj formulář `New_user_role`
- aktér zadá hodnoty všech povinných atributů tabulky `User_role`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `user_role_id`
- po úspěšném ověření a potvrzení aktérem se v tabulce `User_role` vloží nová věta s primárním klíčem `user_role_id`



Obrázek 10: DFD diagram úrovně 1 - správa rolí

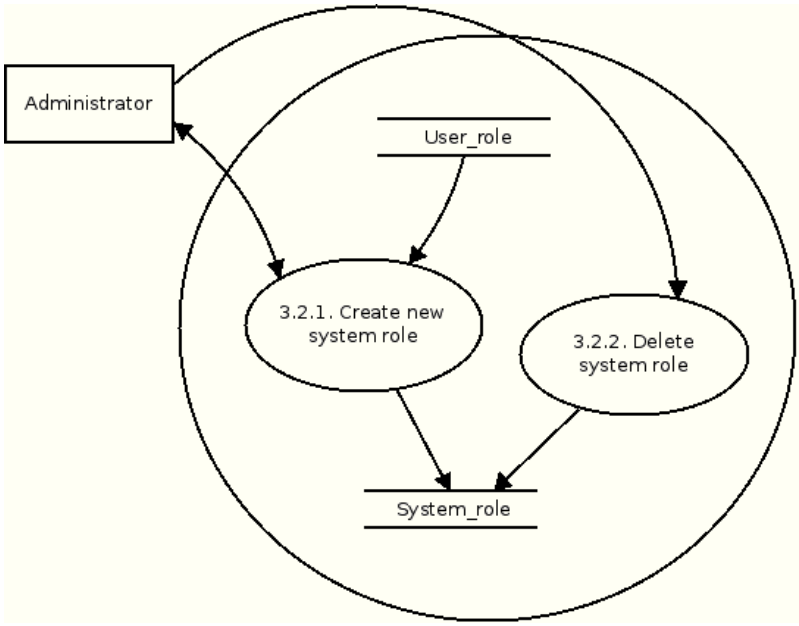


Obrázek 11: DFD diagram úrovně 2 - správa uživatelských rolí

3.3.3.1.2 Smazání uživatelské role

- ověř existenci `user_role_id`
- ověř, že záznam klíče `user_role_id` není použit v ostatních tabulkách

- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam



Obrázek 12: DFD diagram úrovně 2 - správa systémových rolí

3.3.3.2 Správa systémových rolí

3.3.3.2.1 Vytvoření nové systémové role

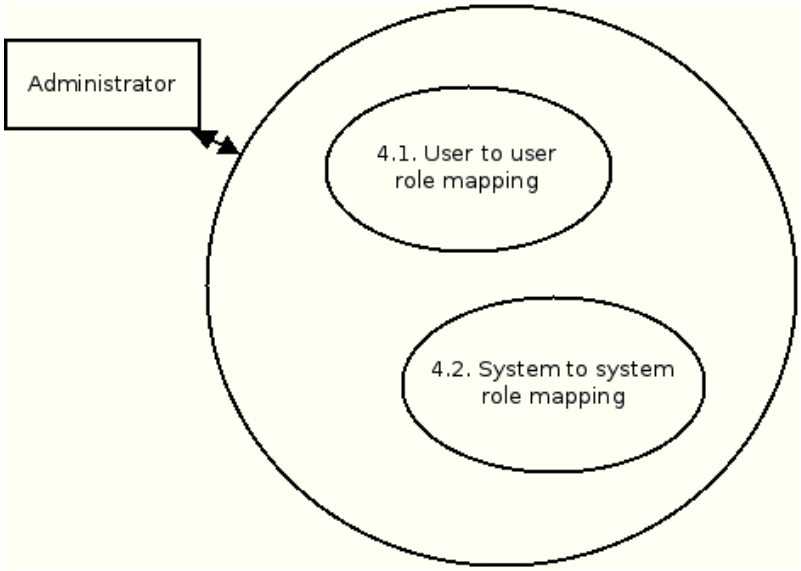
- generuj formulář `New_system_role`
- aktér vybere jednu hodnotu atributu `user_role_id` a zadá hodnoty všech povinných atributů tabulky `User_role`
- ověří se, zda již neexistuje záznam se stejnými atributy
- generuj nové `system_role_id`
- po úspěšném ověření a potvrzení aktérem se v tabulce `System_role` vloží nová věta s primárním klíčem `system_role_id` a cizím klíčem `user_role_id`

3.3.3.2.2 Smazání systémové role

- ověř existenci `system_role_id`
- ověř, že záznam klíče `system_role_id` není použit v ostatních tabulkách

- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.4 Přiřazení do rolí



Obrázek 13: DFD diagram úrovně 1 - přiřazení do rolí

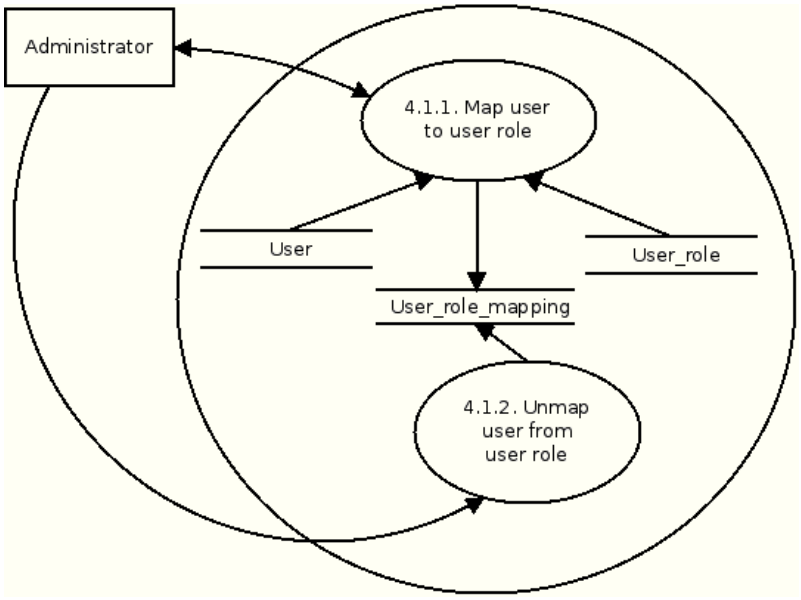
3.3.4.1 Mapování uživatelských rolí

3.3.4.1.1 Přiřazení uživatele do uživatelské role

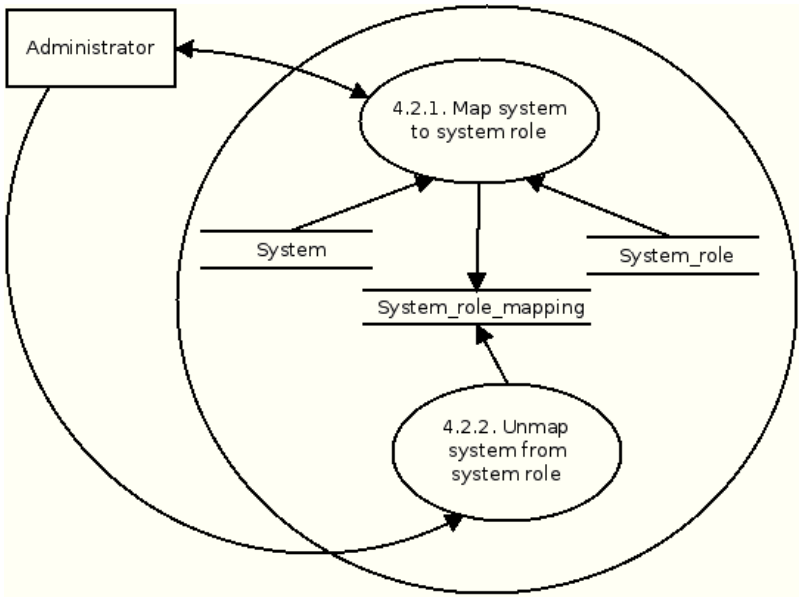
- generuj formulář `Map_user_to_user_role`
- aktér vybere jednu hodnotu atributu `uid` z tabulky `User` a `user_role_id` z tabulky `User_role` a zadá hodnoty všech povinných atributů tabulky `User_role_mapping`
- ověří se, zda již neexistuje záznam se stejnými atributy
- po úspěšném ověření a potvrzení aktérem se v tabulce `User_role_mapping` vloží nová věta s cizími klíči `uid` a `user_role_id`

3.3.4.1.2 Odebrání uživatele z uživatelské role

- ověř existenci záznamu `uid` `user_role_id` v tabulce `User_role_mapping`
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam



Obrázek 14: DFD diagram úrovně 2 - mapování uživatelských rolí



Obrázek 15: DFD diagram úrovně 2 - mapování systémových rolí

3.3.4.2 Mapování systémových rolí

3.3.4.2.1 Přiřazení systému do systémové role

- generuj formulář `Map_system_to_system_role`

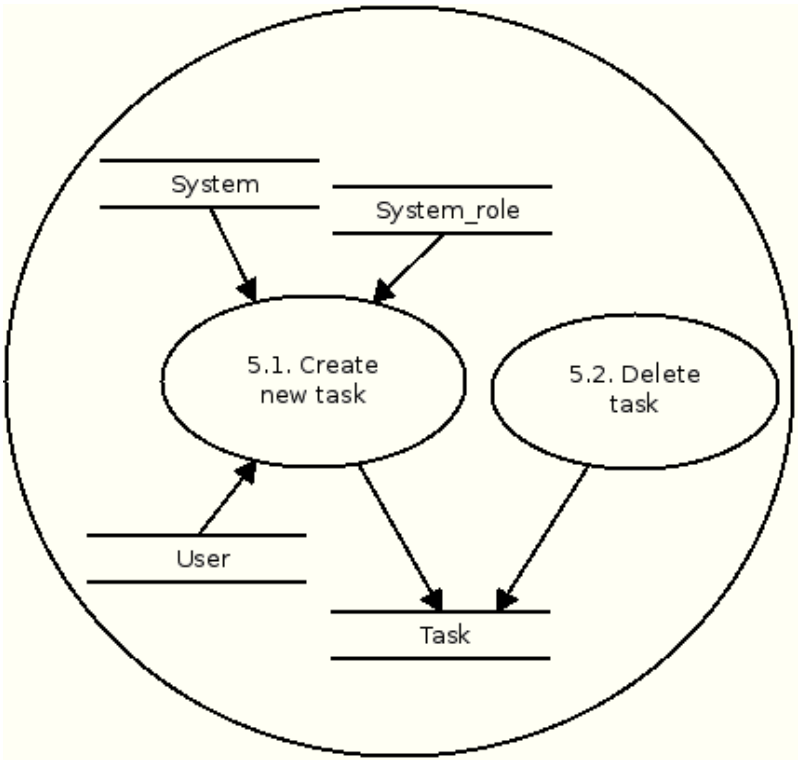
- aktér vybere jednu hodnotu atributu `system_id` z tabulky `System` a `system_role_id` z tabulky `System_role` a zadá hodnoty všech povinných atributů tabulky `System_role_mapping`
- ověří se, zda již neexistuje záznam se stejnými atributy
- po úspěšném ověření a potvrzení aktérem se v tabulce `System_role_mapping` vloží nová věta s cizími klíči `system_id` a `system_role_id`

3.3.4.2.2 Odebrání systému ze systémové role

- ověř existenci záznamu `system_id` a `system_role_id` v tabulce `System_role_mapping`
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.5 Správa úloh

Úlohy nebudou spravovány uživatelem, ale samotným systémem. Jak je možno vidět na obrázku 16, budou bez aktéra.



Obrázek 16: DFD diagram úrovně 1 - správa úloh

3.3.5.1 Vytvoření nové úlohy

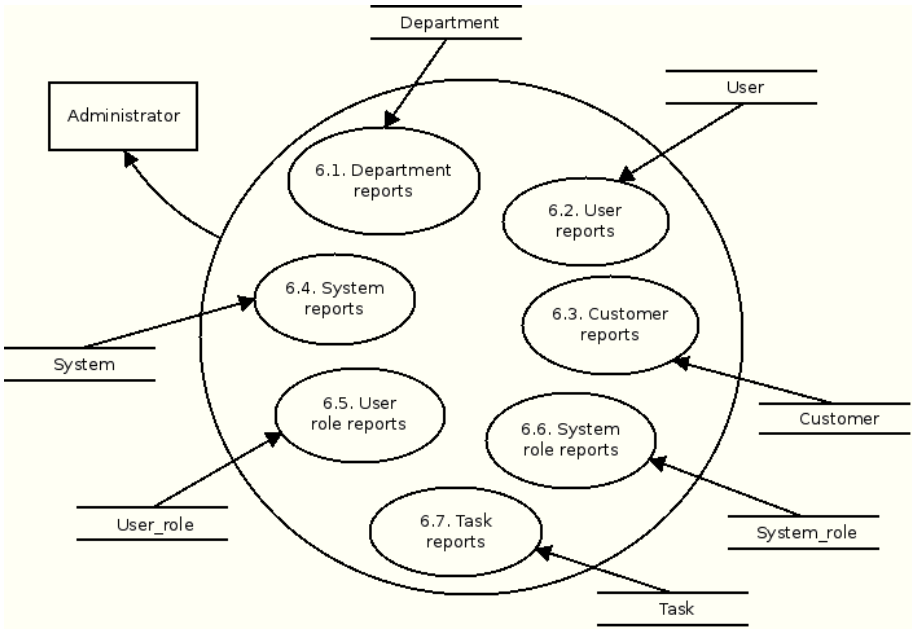
- ověří se, zda již neexistuje záznam s atributy `operation`, `system_id`, nebo také `system_role_id` a `system_id`
- po úspěšném ověření a potvrzení aktérem se v tabulce `Task` vloží nová věta s primárním klíčem `operation`, dále cizími klíči `system_id`, nebo také `system_role_id` a `system_id`

3.3.5.2 Smazání úlohy

- ověř existenci záznamu `operation`, `system_id`, nebo také `system_role_id` a `system_id` v tabulce `Task`
- vypiš dotaz na smazání záznamu
- v případě potvrzení odstraň záznam

3.3.6 Výpisy

Nad všemi tabulkami z obrázku 17 je možnost provádění základních třídění a vyhledávání.



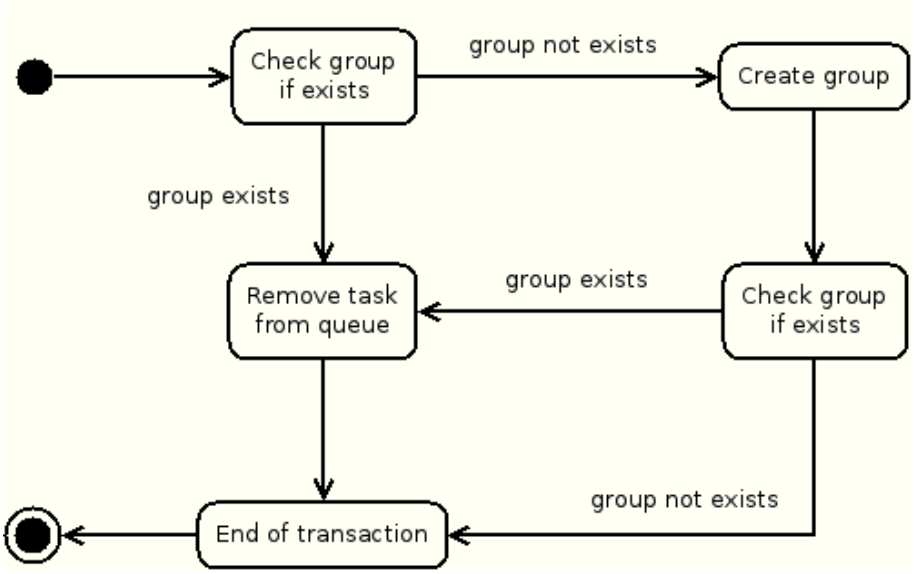
Obrázek 17: DFD diagram úrovně 1 - výpisy

3.4 Časová analýza

Součástí časové analýzy jsou stavové diagramy funkcí a metod. V následujících kapitolách jsou popsány algoritmy implementovaných metod pro správu uživatelů. Máme definovány všechny možné stavy, přechody a události, na které můžeme narazit.

3.4.1 Vytvoření skupiny

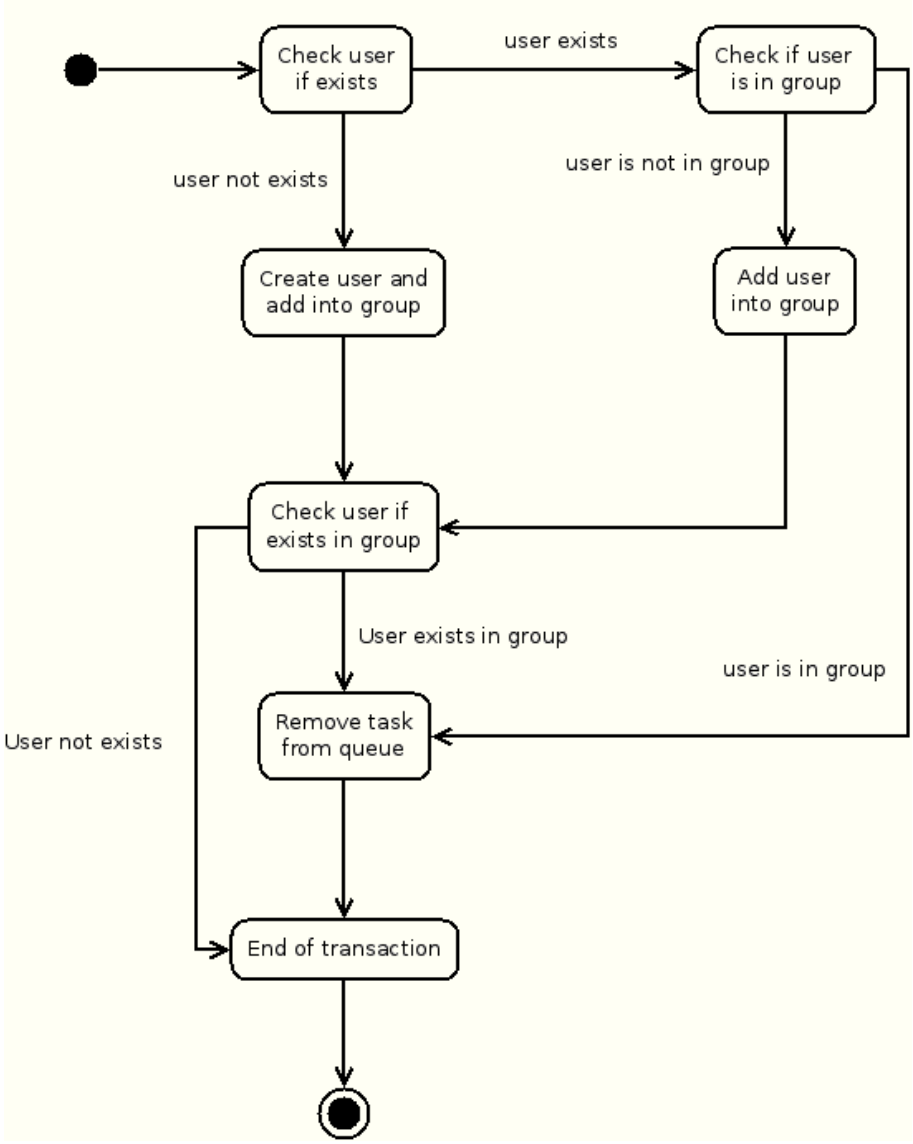
Před vytvořením samotné skupiny musíme ověřit, zda skupina již neexistuje. Při úspěšném ověření existující skupiny můžeme odebrat úlohu z fronty.



Obrázek 18: Stavový diagram přidání skupiny

3.4.2 Vytvoření uživatele, zařazení do skupiny

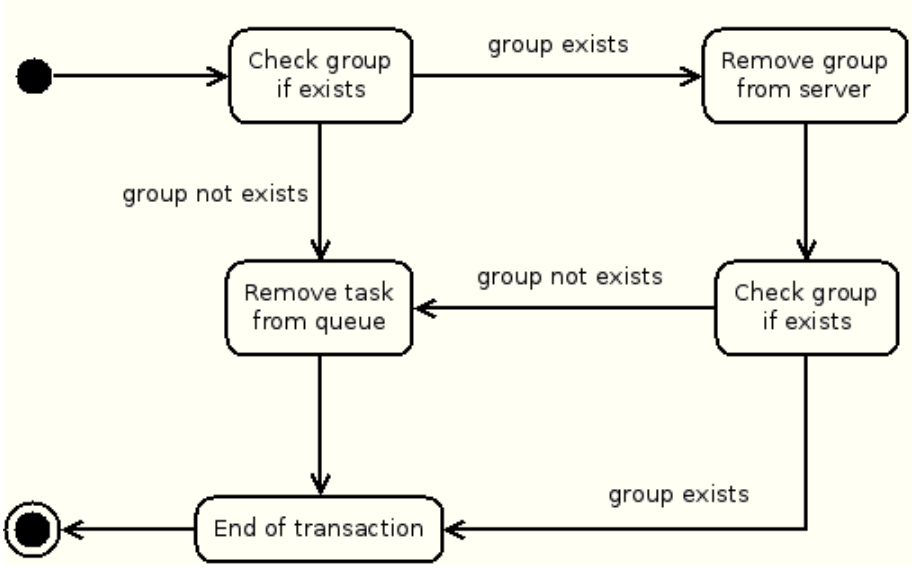
U vytváření uživatele zjišťujeme, zda-li uživatel již existuje. Pokud ano, nebudeme jej vytvářet, pouze ho přidáme do skupiny. V opačném případě jej vytvoříme zároveň se zařazením do skupiny. Po jeho úspěšném ověření můžeme odebrat úlohu z fronty.



Obrázek 19: Stavový diagram přidání uživatele

3.4.3 Odebrání skupiny

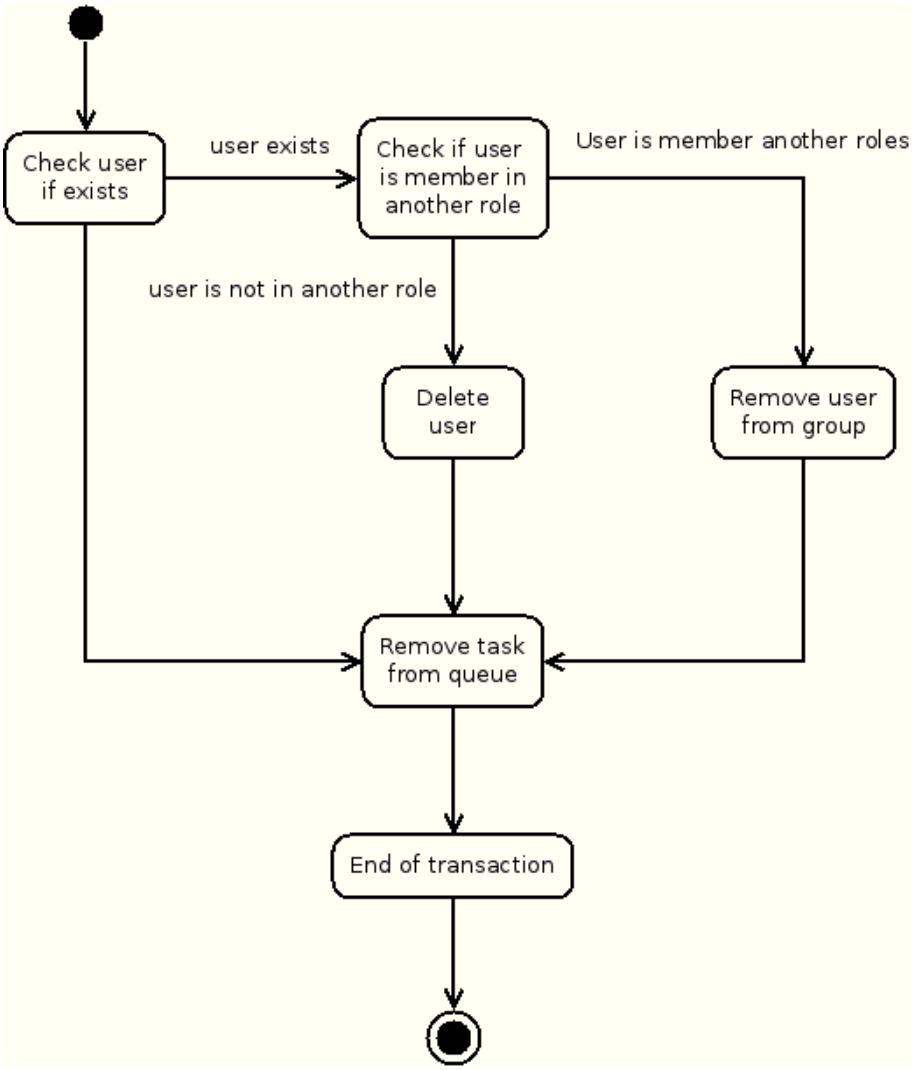
Před samotným smazáním skupiny ověřujeme, zda-li skupina již nebyla odebrána dříve. V případě úspěšného smazáním můžeme úlohu odebrat z fronty.



Obrázek 20: Stavový diagram odebrání skupiny

3.4.4 Odstranění uživatele, odebrání ze skupiny

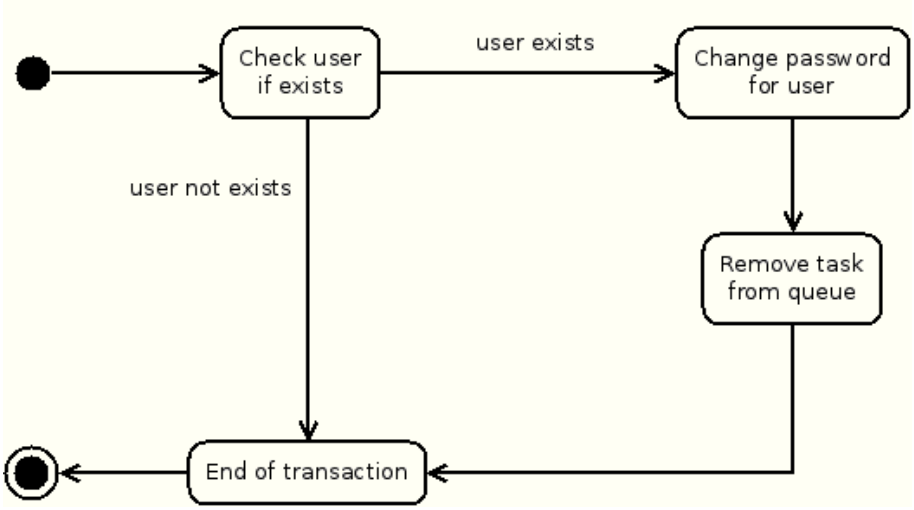
Před odstraněním uživatele se musíme ubezpečit, zda-li není součástí jiných skupin. Pokud ano, odebereme jej jen z příslušné skupiny. Pokud není, můžeme uživatele odstranit. V případě úspěchu můžeme úlohu odebrat z fronty.



Obrázek 21: Stavový diagram odebrání uživatele

3.4.5 Změna hesla uživatele

Před každou změnou hesla zjišťujeme, zda-li uživatel existuje v systému. Pokud ano, můžeme mu změnit jeho heslo a poté můžeme úlohu odebrat z fronty.



Obrázek 22: Stavový diagram změny hesla

4 Implementace

4.1 Požadavky

Hlavní část zdrojového kódu je napsána v interpretovaném programovacím jazyce Perl[10]. Jazyk má volně dostupné moduly třetích stran v CPAN[11]. Všechna dostupná rozhraní a knihovny jsou obsažena v modulech. Seznam všech používaných modulů je popsán v kapitole 5. Program využívá prvky objektového programování a paralelního programování[12].

Všechny zdrojové kódy jsou psané ve znakové sadě `UTF-8`. Pro tvorbu a modelování UML diagramů jsem využil volně dostupný program Dia[13].

Zdrojové kódy aplikace budou perlovské skripty ukládány v souborech zakončených `.pl`, které budou zpracovávat uživatelské vstupy, a dále podle potřeby využívat naprogramované perlovské moduly s příponou `.pm`. Skripty jsou spouštěny klasicky z příkazové řádky. Veškeré konfigurační soubory jsou umístěny v adresáři `/etc/idm`.

4.2 Abstraktní třída IDM

Podle ER diagramu můžeme shlédnout, že je aplikace rozdělena do několika tabulek. Pro každou z nich máme vytvořený modul definující třídu a příslušný spustitelný skript. Přehled všech funkcí je popsán v kapitole 3.1.5. Třídy Department, User, Customer, System, User_role, System_role, User_role_mapping a System_role_mapping mají společnou abstraktní třídu IDM. I když neodpovídá žádnému konkrétnímu objektu, tak nám poskytuje metody, které jsou odvozeným třídám společné. Výhoda je, že nemusíme metody pro každou třídu definovat zvlášť, ale jen jednou. Případná změna kódu se dotkne všech tříd, nemusíme se obávat, že bychom zapomněli dopsat část zdrojového kódu. Všem třídám, které abstraktní třídu využívají, odpovídá stejné schéma objektu. To znamená, že při vytváření objektu zadáváme jméno tabulky, jména atributů primárních klíčů, cizích klíčů a unikátních jmen. Můžeme vyjmenovat několik společných metod abstraktní třídy, a to vytvoření konstruktoru nové třídy, přidání nového objektu do tabulky, vyhledání objektu v tabulce podle klíčů, či unikátních jmen, smazání objektu, apod.

Podstatná většina tříd, využívající abstraktní třídu IDM, neobsahuje žádný speciální kód. Obsahuje pouze přidání nového řádku na databázové úrovní, kde se zjišťuje, zda-li existuje, či ne. Dále se u této funkce pomocí regulárních výrazů ověřují všechny atributy příslušného nového řádku. Až poté se nový řádek vloží do tabulky. Odebrání řádku z tabulky je implementováno v abstraktní třídě IDM, kde se ověřuje, zda-li řádek v tabulce vůbec existuje.

Pak je zde třída System, jejíž metody nepracují pouze na databázové úrovní, ale provádějí také jiné operace mimo databázi. Její podrobný popis je v kapitole 4.6.

[10] Veškeré informace během programování v Perlu jsem použil z knížky Pavla Satrapy[7], Perl pro zelenáče. Dále jsem čerpal z uživatelkých fór a tutoriálů Perlu[12], [13].

[11] Veškerá dokumentace k modulům je rovněž na stránkách CPAN[11].

[12] Pro nastudování problematiky paralelizace procesů bylo čerpáno z materiálů Petra Olivky[8].

[13] Veškerá dokumentace a tutoriál k aplikaci Dia jsou obsaženy na oficiálních webových stránkách projektu[14].

4.3 Nástroj logování

Během implementace, a také při spouštění aplikace, je nutné vypisovat výstupy programu na obrazovku, což se později ukáže jako velmi neefektivní činnost. Abychom zabránili případným nesrovnalostem bylo potřeba zahrnout do aplikace nějaký nástroj pro ladění kódu a analýzu dat. Logovacích nástrojů existuje velké množství a každý má své výhody a nevýhody. V unixovém prostředí se můžeme nejčastěji setkat s nástrojem syslog. V perlu existuje modul `Log::Log4perl`, který je portem velmi populárního `Log4j` logovacího balíčku známého v Javě a syslogu je velmi podobný. Zprávy dokáže generovat podle jednotlivých úrovní, a to buď do souboru, nebo přímo na obrazovku. Konfigurace je možná ve dvou módech, jednoduchý a standardní. Standardní mód byl použit kvůli větší možnosti využití a konfigurovatelnosti. Nastavení logovacího nástroje je zapsáno v souboru `/etc/idm/log.conf`.

4.4 Databázový systém

Datový model aplikace je implementován v prostředí relační databáze SQLite verze 3. Tento databázový systém byl vybrán díky své nenáročnosti. V době, kdy se s databází nepracuje, neběží žádné procesy navíc a celá databáze je uložena v jednom souboru. Pro přístup k databázím je nutná instalace `DBI` modulu v perlu (DataBase Interface). Je to rozhraní, umožňující přístup k nejznámějším databázovým systémům. V případě SQLite databáze instalujeme modul `DBI::SQLite`. Pro podporu cizích klíčů v databázi SQLite je nutno při každé inicializaci databáze spustit příkaz `PRAGMA foreign_keys = ON`. Tato minimalistická databáze se osvědčila hlavně při ladění aplikace během jejího vývoje.

Pro nasazení systému do produkčního prostředí byl také vytvořen modul pro databázový systém PostgreSQL, který je plně kompatibilní s datovým modelem. Je však nutná instalace nového databázového systému a perlovského modulu `DBI::Pg`. Primárně je však pro aplikaci použita databáze SQLite verze 3, jelikož nebude potřeba přistupovat k databázi vzdáleně.

4.5 Úlohy

Celý provoz aplikace je rozdělen do dvou nezávislých úseků. Prvním z nich jsou funkce, ve které se provádějí kroky vykonávané jen čistě nad databázovým systémem. Mezi ně patří například vytváření záznamů uživatelů, zákazníků, rolí, podřízených systémů apod. Dalšími funkcemi jsou takové, které je nutno opět nejprve vykonat nad databázovým systémem. Poté se musí příslušné příkazy provést na podřízených systémech. Jedná se hlavně o přiřazování uživatelů a podřízených systémů do rolí, či změna hesel uživatelů. Tyto funkce jsou zařazeny do druhého úseku. Pro každou takovou úlohu vytvoříme záznam ve speciální tabulce úloh – Task. Z časového hlediska to bude vypadat tak, že se každá funkce nejprve vykoná nad databázovým systémem (součástí je také validace parametrů, čí ověření zda-li již neexistuje stejný záznam) a poté pro funkce, u kterých bude nutné se připojit k podřízeným systémům se vytvoří záznam v tabulce úloh. Správce systému po dokončení všech nastavení a kroků spustí provádění veškerých úloh na podří-

zených systémech. Aplikaci může bezpečně ukončit, protože bude běžet jako nezávislý proces na pozadí. V případě úspěšného vykonání úlohy se provede smazání záznamu z tabulky úloh. Po celou dobu provádění úloh na podřízených systémech je blokováno znovu spustit aplikaci, dokud všechny úlohy nebudou provedeny. Veškeré nedokončené úlohy zůstanou v tabulce úloh, ty pak mohou být například odeslány správci formou emailové zprávy. Správce může vyčíst z logovacích souborů důvod neprovedení úlohy a provést další případné kroky.

4.6 Správa podřízených systémů

Podřízenými systémy jsou myšleny UNIXové operační systémy všech možných platforem, od volně šiřitelných, po komerční. Byly vybrány nejčastěji používané systémy – nekomerční Linux a z komerčních AIX, Solaris a HP-UX. Pro každou platformu jsou uvedeny příkazy nutné ke správě účtů. Pokud bychom chtěli do aplikace zařadit systémy ostatních platforem (BSD, True64, Mac OS, apod.), je nutno pouze uvést tyto příkazy.

Při vytváření záznamu nového systému do tabulky podřízených systémů, musíme specifikovat několik atributů, které nám slouží pro identifikaci systému. Tyto informace jsou pak použity při následném připojení do systému:

- `platform` - identifikace platformy operačního systému (např. SunOS, AIX, Linux, HP-UX)
- `dhome` - kořenová cesta k domovským adresářům vytvářených uživatelů
- `shell` - cesta standardního interpretu příkazů pro nově vytvořené uživatele
- `protocol` - typ protokolu připojení do systému (ssh, telnet, apod.)
- `port` - číslo portu TCP/IP protokolu
- `account_type` - specifikace typu účtu přihlášení do systému, viz také kapitola 4.8
- `account_name` - jméno účtu pro přihlášení do systému, viz také kapitola 4.8
- `database_version` - specifikace verze databáze hesel, viz také kapitola 4.8.1
- `last_jumppoint` - identifikace předcházejícího systému, viz také kapitola 4.8.3
- `jumppoint_flag` - příznak jumppointu, viz také kapitola 4.8.3

4.7 Expect

Jedním ze základních požadavků aplikace byla implementace bez použití centralizovaného úložiště. Zajistíme to jedině tak, že všechny účty budou vytvářeny lokálně. Abychom se dostali do konkrétního systému a mohli v něm vykonat příslušné příkazy, je nutné se k němu nejprve korektně připojit. Jako první varianta se nabídla využít již implementovaných perlovských modulů `Net::SSH` a `Net::Telnet` pro připojení do systému.

Problém však nastává v situaci, kde se není možno na daný server připojit přímo, ale přes více jiných systémů. Proto jsem hledal jinou vhodnou alternativu v podobě interaktivní komunikace vstupu, výstupu. Podařilo se najít perlovský modul Expect v CPAN, který umožňuje vytvořit v terminálu nový proces, případně se připojit na stávající pseudoterminál a v něm provádět interaktivní úlohy bez účasti operátora. Pomocí něj se na terminál vyšle příkaz, ten poté očekává nějaký výsledek, na který určitým způsobem reaguje. Pokud ale neobdrží nic do předem stanoveného časového intervalu, proces opět zareaguje podle stanovených funkcí. Pomocí modulu Expect je možno se přihlašovat do systémů a vykonávat dané příkazy.

Pro všechny tyto funkce slouží modul Expect_common, ve kterém jsou funkce pro vytvoření nového pseudoterminálu, připojení do něj, připojení do systému, odpojení ze systému, funkce pro provádění příkazů, změna hesla uživatele a interaktivní mód.

```
use Expect;
# create an Expect object by spawning another process
my $exp = Expect->spawn($command, @params)
   or die "Cannot spawn $command: $!\n";

# or by using an already opened filehandle (e.g. from Net::Telnet)
my $exp = Expect->exp_init(\*FILEHANDLE);

# if you prefer the OO mindset:
my $exp = new Expect;
$exp->raw_pty(1);
$exp->spawn($command, @parameters)
   or die "Cannot spawn $command: $!\n";

# send some string there:
$exp->send("string\n");

# or, for the filehandle mindset:
print $exp "string\n";

# then do some pattern matching with either the simple interface
$patidx = $exp->expect($timeout, @match_patterns);

# or multi-match on several spawned commands with callbacks,
# just like the Tcl version
$exp->expect($timeout,
   [ qr/regex1/ => sub { my $exp = shift;
      $exp->send("response\n");
      exp_continue; } ],
   [ "regexp2" , \&callback, @cbparms ],);

# if no longer needed, do a soft_close to nicely shut down the command
$exp->soft_close();

# or be less patient with
$exp->hard_close();
```

Výpis 1: Ukázka použití modulu Expect

4.7.1 Problém paralelismu v Expectu

Připojení k jednotlivým systémům je možno provádět sériově za sebou, ale kdybychom chtěli vytvářet účty na stovce systémech, tak se nám doba vykonávání prodlouží také stokrát. Proto byla časová souběžnost (paralelismus) procesů jedním z dalších základních požadavků při návrhu aplikace.

U modulu Expect se můžeme dočíst, že vytváření vláken a procesů je možno, ale s určitými omezeními. Metoda `spawn()`, v níž se vytváří nový pseudoterminál, musí být volána přímo v hlavním vlákně, nebo před rozvětvením procesu. Je to z toho důvodu, že samotná metoda `spawn()` již používá funkce pro vytváření vláken a nových procesů a není navržena tak, abychom ji volali v potomcích vláken a procesů. Metodu `spawn()` jsem proto použil ze zdrojového kódu modulu Net::Telnet, kterou následně šlo volat v již vytvořených vláknech a procesech. Po zavolání modifikované metody `spawn()` se volá metoda `Expect->exp_init(\*FILEHANDLE)` pro použití právě vytvořeného pseudoterminálu.

```
sub spawn {

    my($cmd) = @_;
    my($pid, $pty, $tty, $tty_fd);

    ## Create a new pseudo terminal.
    $pty = new IO::Pty
        or die $!;

    ## Execute the program in another process.
    unless ($pid = fork) {  # child process
        die "problem spawning program: $!\n" unless defined $pid;

        ## Disassociate process from existing controlling terminal.

        use POSIX ();
        POSIX::setsid
            or die "setsid failed: $!";

        ## Associate process with a new controlling terminal.
        $tty = $pty->slave;
        $pty->make_slave_controlling_terminal();
        $tty_fd = $tty->fileno;
        close $pty;

        ## Make stdio use the new controlling terminal.
        open STDIN, "<&$tty_fd" or die $!;
        open STDOUT, ">&$tty_fd" or die $!;
        open STDERR, ">&STDOUT" or die $!;
        close $tty;

        ## Execute requested program.
        exec "sh"
        #exec $cmd
            or die "problem executing sh\n";
```

```
    } # end child process

    return $pty;
} # end sub spawn
```

Výpis 2: Ukázka převzaté metody spawn() z modulu Net::Telnet

4.7.2 Zavedení časové prodlevy

Jak je zmíněno výše, po vyslání příkazu na pseudoterminál pomocí metody `send()` zavoláme metodu `expect()`, ve které zachytáváme očekávaný výstup. Během zavedení paralelismu procesů se stávalo, že se na terminál poslal příkaz metodou `send()`. Metoda již byla ukončena na lokálním počítači, ale samotný příkaz ještě nebyl na vzdáleném stroji spuštěn a mezitím se již očekával daný výstup. Metoda `expect()` zachytila výstup o krok dříve než měla. O podobném problému je zmíněno v samotné dokumentaci Expectu, kde doporučují mezi prováděním příkazu (`spawn()`, `send()`) a jeho zachycením(`expect()`) přidat jednosekundovou prodlevu pomocí příkazu `sleep 1`. Poté se chyba v přeskakování jednotlivých kroků již nevyskytovala.

4.8 Popis metody připojení do systému

Abychom mohli spouštět příslušné příkazy pro správu uživatelů na systémech, musíme se do systému přihlásit pod privilegovaným uživatelem pro spuštění těchto příkazů. Během přidávání nového systému do databáze nastavujeme všechny potřebné atributy potřebné pro připojení. V první řadě se jedná o protokol, a to buď program telnet, nebo SSH. U každého protokolu uvádíme číslo TCP/IP portu. Standardně je pro SSH port číslo 22 a pro telnet port číslo 23. Během připojení musíme rovněž uvést jméno uživatele, přes kterého se budeme autentizovat. Buď se bude jednat o účet správce, nebo bude možno použít jméno účtu z databáze hesel, viz kapitola 4.8.1.

Dále musíme specifikovat jakým způsobem se dostaneme na privilegovaného uživatele. Máme několik možností:

- `system account + su` - systémový učet a heslo uživatele root, oba z databáze hesel
- `system account + sudo` - systémový účet, včetně přihlášení na roota přes sudo
- `personal account + su` - osobní účet správce systému a root z databáze hesel
- `personal account + sudo` - osboní účet, včetně přihlášení na roota přes sudo
- `direct root` - přímý root a jeho heslo z databáze hesel

4.8.1 Databáze hesel

Databáze hesel jsou speciální informační systémy, které obsahují jména systémů a jejich přihlašovací údaje. Přístup k nim je zabezpečen přihlašovacími údaji a nastavenými právy. Pro zjišťování přihlašovacích údajů k podřízeným systémům jsou vytvořeny moduly First.pm a Second.pm. Moduly pracují tak, že jim podložíme doménové jméno systému a jméno uživatele, a pak nám vrátí zjištěné heslo. Oba moduly používají perlovský modul `WWW::Mechanize`, který interaktivně prohlíží a analyzuje webové stránky a vyplňuje webové formuláře.

4.8.2 Modul Security

Pokud bychom chtěli pro připojení použít účet správce, místo zjišťování přihlašovacích údajů z databáze hesel, musíme jej načíst ze zašifrovaného textového souboru. Pro jeho dešifrování je vytvořená metoda Security.pm, která používá modul `Crypt::CBC` pro šifrování a dešifrování pomocí symetrického klíče. Ten je po nás dotázán na začátku při spuštění aplikace. Modul Security.pm se nepoužívá jen pro uložení přihlašovacích údajů správce systému, ale také pro přihlašovací údaje databází hesel uživatelů.

4.8.3 Popis jumppoint systému

Posledními dvěma položkami jsou informace o předchozím jumppointu a jestli má daný systém příznak jumppointu. Jumppoint je prostředníkem mezi řídicím a spravovaným podřízeným systémem. Jelikož takových prostředníků může být několik za sebou, tak i prostředník může mít svůj předchozí jumppoint. Pokud je u podřízeného systému nastavený příznak jumppointu, tak se aplikace nebude snažit připojit přímo na privilegovaného uživatele, ale jen na prvního, na kterého mu je umožněno. Všechny jumppointy jsou spravovány manuálně, z bezpečnostního hlediska není povolen přístup na roota.

4.9 Provádění požadavků po připojení

Po úspěšném připojení k systému jsou připraveny k vykonání veškeré požadavky z tabulky úloh. Všechny tyto požadavky jsou funkcemi při zařazování uživatelů a podřízených systémů do rolí. Pokud například přiřadíme systém do role a tato role je provázána s uživateli, musíme v systému vytvořit novou skupinu včetně uživatelů. Chceme-li například změnit heslo uživatele, vyhledáme všechny systémy, ve kterých figuruje. Popis jednotlivých kroků a stavů funkcí je popsán v časové analýze, kapitola 3.4. Jejich pořadí, ve kterém se postupně provádějí na každém systému je následující:

- `groupadd` - vytvoření skupin
- `useradd` - vytvoření uživatelů
- `passwdchg` - změna hesel uživatelů
- `userdel` - odebrání uživatelů

- `groupdel` - odebrání skupin

4.10 Uživatelské rozhraní

Součástí této práce je také textové uživatelské rozhraní, které slouží k zadávání údajů, výpisů a řídí veškerou komunikaci s aplikací. Po spuštění příkazového interpretru se nám vytvoří prostředí příkazové řádky. Spouštěcí skripty voláme tak, že zadáme název skriptu a za něj doplníme parametry, které slouží pro spuštění příkazu v aplikaci. Zde je seznam všech spouštěcích skriptů, podrobný popis parametrů je vypsán v příloze A uživatelské dokumentace.

- `department.pl` - správa oddělení
- `user.pl` - správa uživatelů
- `customer.pl` - správa zákazníků
- `system.pl` - správa systémů
- `user_role.pl` - správa uživatelských rolí
- `system_role.pl` - správa systémových rolí
- `user_map.pl` - mapování uživatelů
- `system_map.pl` - mapování systémů
- `task.pl` - spouštění úloh
- `security.pl` - pomocný skript pro šifrování textu (uživatelská jména, hesla, apod.)

5 Postup nasazení systému

V této kapitole jsou popsány všechny nutné předpoklady pro zprovoznění systému na Linuxovém stroji. Já sám jsem systém vyzkoušel na distribuci Fedora a poté na jeho stabilní odnoži CentOS 6.

5.1 Instalace Perlu a modulů

Jelikož je aplikace napsána v programovacím jazyce Perlu je nutné mít nainstalované jeho balíčky. Nejjednodušší je instalace binárních balíčků z balíčkovacího systému dané distribuce. Dalším předpokladem je instalace Perlovských modulů, které můžeme stáhnout z CPAN serveru. Pokud naše distribuce obsahuje tyto balíčky v repozitáři, je jednodušší variantou instalace pomocí balíčkovacího systému. Aplikace používá tyto následující moduly[14]:

- `Log::Log4perl` - implementace logovacího modulu Log4j pro perl
- `DBI` - databázové nezávislé rozhraní v perlu
- `DBD::SQLite` - RDBMS v DBI ovladači
- `Crypt::CBC` - šifrování dat pomocí CBC módu
- `WWW::Mechanize` - interaktivní programové procházení webů
- `WWW::Mechanize::Frames` - podpora rámců
- `HTTP::Cookies` - podpora cookies
- `HTML::Entities` - šifrování, dešifrování řetězců s HTML entitami
- `Expect` - provádění interaktivních úloh bez účasti operátora
- `IO::Tty` - vytváření pseudoterminálů
- `Data::Dumper` - formátování tisku datových struktur
- `Parallel::ForkManager` - Jednoduchý manažer zpracování paralelních procesů
- `Crypt::Blowfish` - Blowfish šifrovací modul
- `Getopt::Long` - podpora dlouhých přepínačů při zpracování parametrů příkazové řádky

[14] Veškerá dokumentace k modulům je rovněž na stránkách CPAN[11].

5.2 Nastavení konfiguračních souborů

Součástí aplikace jsou také konfigurační soubory, které je nutno zkopírovat do adresáře `/etc/idm`. Logovácí soubory aplikace jsou také umístěny ve vyhrazeném adresáři `/var/idm`. Oba adresáře musíme vytvořit a nastavit přístupová práva pro vlastníka aplikace:

```
[root@CentOS /]# mkdir /var/idm
[root@CentOS /]# chown -R idm:idm /var/idm
[root@CentOS /]# chmod -R 600 /var/idm

[root@CentOS /]# mkdir /etc/idm
[root@CentOS /]# chown -R idm:idm /etc/idm
[root@CentOS /]# chmod -R 600 /etc/idm
```

Výpis 3: Vytvoření a nastavení práv adresářů na řídicím serveru.

5.2.1 Šifrování přihlašovacích údajů

Pro přihlášení na podřízené systémy se používají přihlašovací údaje vytažené buď z databáze hesel, nebo přímé účty. Zašifrované údaje jsou uloženy v souboru `/etc/idm/config` pomocí symetrické šifry Blowfish:

```
$::credentials = {
    'FIRST_DB' => {
        'url'      => 'encrypted_url',
        'username' => 'encrypted_username',
        'password' => 'encrypted_password',
    },
    'SECOND_DB' => {
        'url'      => 'encrypted_url',
        'username' => 'encrypted_username',
        'password' => 'encrypted_password',
    },
    'JUMP' => {
        'username' => 'encrypted_username',
        'password' => 'encrypted_password',
    },
};
```

Výpis 4: Ukázka konfiguračního souboru /etc/idm/config

5.2.2 Logovací soubory

Na následujícím kódu je ukázka konfiguračního souboru `/etc/idm/log.conf` pro logování veškerých zpráv aplikace. V něm je možno nastavit jak logování na obrazovku, tak do souboru. Pro každý vytvořený `.pm` modul je logování iniciováno zvlášť.

```
############################################################
# Configuration of a logger with Log::Log4perl
# file appender      LOGFILE
# screen appender SCREEN
############################################################

layout_class              = Log::Log4perl::Layout::PatternLayout
 file_pattern             = %d{yyyy-MM-dd HH:mm:ss} "%c" %p [%F:%L]: %m%n
screen_pattern            = %p level '%c' [%F:%L]: %m%n

log4perl.rootLogger       = WARN, SCREEN, LOGFILE

log4perl.appender.LOGFILE  = Log::Log4perl::Appender::File
log4perl.appender.LOGFILE.filename = /var/idm/error.log
log4perl.appender.LOGFILE.layout = ${layout_class}
log4perl.appender.LOGFILE.layout.ConversionPattern = ${file_pattern}

log4perl.appender.SCREEN  = Log::Log4perl::Appender::Screen
log4perl.appender.SCREEN.stderr  = 0
log4perl.appender.SCREEN.layout  = ${layout_class}
log4perl.appender.SCREEN.layout.ConversionPattern = ${screen_pattern}
```

Výpis 5: Ukázka konfiguračního souboru loggeru Log::Log4Perl.

Následuje ukázka výpisu logu:

```
[root@CentOS /]# tail /var/idm/error.log
2012-04-09 14:24:08 "IDM.User_role" WARN [IDM/User_role.pm:39]: User role name unix exists,
    please rename it
2012-04-09 14:24:08 "IDM.User_role" WARN [IDM/User_role.pm:39]: User role name database
    exists, please rename it
2012-04-09 14:24:08 "IDM.System_role" WARN [IDM/System_role.pm:70]: System role name
    vsbdb exists, please rename it
2012-04-09 14:24:08 "IDM.System_role" WARN [IDM/System_role.pm:70]: System role name
    vsbapp exists, please rename it
2012-04-09 14:24:08 "IDM.System" WARN [IDM/System.pm:438]: No tasks exists, exiting.
2012-04-09 14:24:08 "main" INFO [main.pl:223]: Tasks are done
```

Výpis 6: Ukázka logu /var/idm/error.log.

5.3 Nastavení databázového systému

Datový model je postaven nad databázovým systémem SQLite. Pro zprovoznění aplikace je potřeba vytvořit databázi do souboru `/var/idm/database.dbm`. Vytvoření databázového schématu (viz. příloha C) ze souboru `sql/table.sql` a nastavení přístupových práv je následující:

```
[root@CentOS /]# sqlite3 /var/idm/database.dbm < sql/table.sql
[root@CentOS /]# chown -R idm:idm /var/idm/database.dbm
[root@CentOS /]# chmod -R 600 /var/idm/database.dbm
```

Výpis 7: Vytvoření a nastavení práv databázového souboru.

6 Závěr

Ve své práci jsem se snažil navrhnout a vytvořit aplikace pro správu účtů na UNIXových systémech. Zvolil jsem takový způsob implementace, aby aplikace byla v budoucnu snadno rozšiřitelná o další komponenty a vývoj aplikace nemusel být znovu vyvíjen od úplného začátku. Návrh programu byl jeden z nejdůležitějších součástí vývoje této práce, protože od něj se odvíjel celý průběh implementace systému.

Na začátku jsem řešil problém, jak navrhnout takovou aplikaci, která bude nezávislá na platformě daného operačního systému. Jedním z prvních kroků byl návrh vytvoření serverové a klientské části, kde by se klient implementoval a kompiloval zvlášť pro každou používanou platformu. Na hostitelském podřízeném systému by byl poté spuštěný démon klienta pod privilegovaným uživatelem. Klient by byl obsluhován serverem, který by jemu posílal příslušné příkazy pro správu uživatelských účtů. Od této alternativy jsem poté upustil z hlediska složitosti celého problému a komplikací spojených se závislostmi a překladem zdrojových kódů na všech platformách. Snažil jsem se najít jinou, jednodušší alternativu, ve které bych mohl použít stávající knihovny pro správu uživatelů a skupin. Po objevení modulu Expect, který umožňuje interaktivní automatizované spouštění úloh, jsem byl pevně přesvědčený, že to je ta správná cesta, na které bude stát celá aplikace.

Jedním ze základních požadavků na návrh aplikace bylo paralelní zpracování operací. Návrh je pouze teorie, implementace nebyla tak jednoduchá, jak se předpokládalo. Nejen, že bylo nutné přepsat část zdrojového kódu Expectu, ale během testování paralelismu úloh se objevily menší komplikace. Proto byla tato část implementace nejsložitější.

Testování aplikace bylo provedeno na několika UNIXových platformách, a to Linux, AIX a Solaris. Během něj jsem nenarazil na žádný kritický problém, který by mohl ohrozit hostitelské systémy. V budoucnu se počítá s nasazením systému do produkčního prostředí. V první fázi se bude týkat testování u jednoho zákazníka s postupným rozšířením do dalších prostředí.

Jelikož je systém velmi pružný, tak budu se v rámci své pracovní doby věnovat jeho dalšímu rozvoji. Prvním rozšířením bude vytvoření webového rozhraní v MVC architektuře Perl webového frameworku Catalyst. Vzhledem k tomu, že stávající architektura umožňuje přidávat další moduly a funkce, chtěl bych o ně systém dále rozšířit. Jednoduchým příkladem může být funkce pro hromadné ověřování stavu diskových cest na FC řadičích, která by nám ve firmě byla velmi užitečná. Pravidelně nastávají situace, že se aktualizuje firmware na FC switchích, a někdy se cesta po výpadku nevrátí do původního stavu.

Václav Bortlík

7 Reference

[1] BENÁK, Karel. *Použití adresářových služeb v informačních systémech*, Diplomová práce, ČVUT Praha, FSv, 2004.

[2] HANÁČEK, Petr, STAUDEK, Jan. *Správa identity*, Sborník konference DATAKON 2005, Brno, CZ, MUNI, 2005, p. 123-146, ISBN 80-210-3813-6.

[3] EISLER, Mike, LABIAGA, Ricardo, STERN, Hal. *Managing NFS and NIS, 2nd Edition*, O'Reilly, July 2001, ISBN 1-56592-510-6.

[4] CARTER, Gerald. *LDAP System Administration*, O'Reilly, March 2003, ISBN 1-56592-491-6.

[5] GARMAN, Jason. *Kerberos: The Definitive Guide*, O'Reilly, August 2003, ISBN 0-596-00403-6.

[6] ŠARMANOVÁ, Jana. *Databázové a informační systémy*, VŠB-TU Ostrava, FEI, 2008, ISBN 978-80-248-1499-5.

[7] SATRAPA, Pavel. *Perl pro zelenáče*, Neocortex, 2001, ISBN 80-86330-02-8.

[8] OLIVKA, Petr. *Operační systémy*, VŠB-TU Ostrava, FEI, http://poli.cs.vsb.cz/edu/osy/ .

[9] RAY, Deborah, S., RAY, Eric, J.. *Unix podrobný průvodce*, Grada, 2009, ISBN 978-80-247-2125-5.

[10] Linux manuálové stránky, klíčová slova "passwd", "shadow", "group", "telnet", "ssh", "perl", "perlpod".

[11] Dokumentace k Perl modulům, Comprehensive Perl Archive Network, http://www.cpan.org/ .

[12] Informace z prostředí Perlu, tutoriály, uživatelské fórum, http://www.perlmonks.org/ .

[13] Podrobný tutoriál Perlu, http://www.linuxsoft.cz/ .

[14] Dokumentace k Dia, http://projects.gnome.org/dia/docs.html .

A Uživatelská dokumentace

```
[user@host]$ perl department.pl
Usage:
   department.pl -add -depname unix123 -keydir "/etc/idm/pub_keys/unix123" [ -comment "any_
       comment"] | -remove -depname unix123 | -list_all

  -depname - Required. Please give Department name.
  -keydir  - Optional. Root directory for user public keys.
  -comment - Optional. Any comment.
  -add     - Optional. Create department in database.
  -remove  - Optional. Remove department from database.
  - list_all - Optional. List all deparments.

For example:
 Creating department in database:
  [user@host]$ department.pl -add -depname unix123 -keydir "/etc/idm/pub_keys/unix123" -
      comment "any_comment"
 Remove department from database:
  [user@host]$ department.pl -remove -depname unix123
 List all deparments in database:
  [user@host]$ department.pl -list_all
```

Výpis 8: Spouštěcí skript správy oddělení

```
[user@host]$ perl user.pl
Usage:
   user.pl -add -login user123 -depname unix123 -first_name "Johnny" -last_name "Depp" [ -
       password_hash "#HASH#" -phone "420123456789" -email "johnny.depp@abc.com" -
       priv_key "/etc/idm/pub_keys/unix123/user123" -comment "any_comment" ] | -remove -
       login user123 | -list_all | -password -login user123

  -login         - Required. Please give Login name.
  -depname       - Required. Please give Department name.
  -first_name    - Required. Please give First name.
  -last_name     - Required. Please give Last name.
  -password_hash - Optional. User password hash.
  -phone         - Optional. User mobile phone.
  -email         - Optional. User email address.
  -priv_key      - Optional. User private key.
  -comment       - Optional. Any comment.
  -add           - Optional. Create user in database.
  -remove        - Optional. Remove user from database.
  - list_all     - Optional. List all users.
  -password      - Optional. Change user password.

For example:
 Creating user in database:
  [user@host]$ user.pl -add -login user123 -depname unix123 -first_name "Johnny" -last_name
      "Depp" -password_hash "#HASH#" -phone "420123456789" -email "johnny.depp@abc.
      com" -priv_key "/etc/idm/pub_keys/unix123/user123" -comment "any_comment"
 Remove user from database:
  [user@host]$ user.pl -remove -login user123
```

```
 List  all  users in  database:
  [user@host]$ user.pl − list_all
 Change user password:
  [user@host]$ user.pl −password −login user123
```

Výpis 9: Spouštěcí skript správy uživatelů

```
[user@host]$ perl customer.pl
Usage:
   customer.pl −add −customer_name csm123 [ −patrol_name ”patrol” −private_account −
        comment ”any_comment” ] | −remove −customer_name csm123 | −list_all

  −customer_name − Required. Please give customer name.
  −patrol_name    − Optional. Please give patrol  name..
  −private_account − Optional. Please choose if customer has private accounts for users.
  −comment        − Optional. Any comment.
  −add            − Optional. Create new customer in database.
  −remove         − Optional. Remove customer from database.
  − list_all      − Optional. List  all  customers.

For example:
 Creating new customer in database:
  [user@host]$ customer.pl −add −customer_name csm123 [ −patrol_name ”patrol” −
       private_account −comment ”any_comment”
 Remove customer from database:
  [user@host]$ customer.pl −remove −customer_name csm123
 List  all  customers in database:
  [user@host]$ customer.pl −list_all
```

Výpis 10: Spouštěcí skript správy zákazníků

```
[user@host]$ perl system.pl
Usage:
   system.pl −add −fqdn system123 −csm csm123 −platform ”SunOS” −dhome ”/opt/home” −
        shell ”/bin/bash” −protocol ssh −port 22 −account_type SYSTEM_SU [ −account_name ”
        abc123” −db ”FIN” −jumppoint system124 −is_jump −comment ”any_comment” ] | −
        remove −fqdn system123 | −list_all

  −fqdn          − Required. Please give fully  qualified  domain name.
  −csm           − Required. Please give customer name.
  −platform      − Required. Please give platform, you can choose from:
       SunOS          − for Solaris  systems.
       AIX            − for AIX systems.
       HP−UX          − for HP−UX systems.
       Linux          − for Linux systems.

  −dhome         − Required. Please give path for home directories.
  −shell         − Required. Please give shell (command interpret), you can choose from:
       /bin/bash
       /bin/sh
       /usr/bin/ksh
       /usr/bin/bash
       /usr/local/bin/bash
```

```
−protocol      − Required. Please give protocol, you can choose from:
    ssh            − SSH protocol
    telnet         − telnet protocol

−port          − Required. Please give protocol port number, you can choose from:
    22             − default port for SSH
    23             − default port for telnet

−account_type − Required. Type of account, you can choose from:
    SYSTEM_SU    − system account with root password, both from database
    SYSTEM_SUDO − system account with sudoers rights, system account from database
    PERSONAL_SU − personal account with root password from database
    PERSONAL_SUDO − personal account with sudoers rights
    DIRECT_ROOT − direct root account from database
    IDM_SU       − second personal account with root password from database
    IDM_SUDO     − second personal account with sudoers rights

−account_name − Optional. Login name for connection to server.
−db            − Optional. User password database for non−personal accounts, you can choose
    from:
    SWE            − swe database
    FIN            − fin database

−jumppoint     − Optional. Last jumppoint FQDN.
−is_jump       − Optional. Use if system is also jumppoint.
−comment       − Optional. Any comment.
−add           − Optional. Create system in database.
−remove        − Optional. Remove system from database.
− list_all     − Optional. List all systems.

For example:
 Creating system in database:
  [user@host]$ system.pl −add −fqdn system123 −csm csm123 −platform "SunOS" −dhome "/opt
      /home" −shell "/bin/bash" −protocol ssh −port 22 −account_type SYSTEM_SU [ −
      account_name "abc123" −db "FIN" −jumppoint system124 −is_jump −comment "any␣
      comment"
 Remove system from database:
  [user@host]$ system.pl −remove −fqdn system123
 List all systems in database:
  [user@host]$ system.pl −list_all
```

Výpis 11: Spouštěcí skript správy systémů

```
[user@host]$ perl user_role.pl
Usage:
   user_role.pl −add −user_role_name unix_role_123 [ −comment "any␣comment"] | −remove −
       user_role_name unix_role_123 | −list_all

−user_role_name − Required. Please give User role name.
−comment       − Optional. Any comment.
−add           − Optional. Create user role in database.
−remove        − Optional. Remove user role from database.
− list_all     − Optional. List all user roles.
```

```
For example:
 Creating user role in database:
  [user@host]$ user_role.pl -add -user_role_name unix_role_123 -comment "any comment"
 Remove user role from database:
  [user@host]$ user_role.pl -remove -user_role_name unix_role_123
 List all user roles in database:
  [user@host]$ user_role.pl - list_all
```

Výpis 12: Spouštěcí skript správy uživatelských rolí

```
[user@host]$ perl system_role.pl
Usage:
    system_role.pl -add -system_role_name app_systems -user_role_name unix_role_123 -gid
        20000 -group_name app_sys -passwd_comment "Application systems" [ -comment "any
        comment"] | -remove -system_role_name app_systems | -list_all

  -system_role_name - Required. Please give system role name.
  -user_role_name - Required. Please give user role name.
  -gid              - Required. Please give group ID.
  -group_name       - Required. Please give group name.
  -passwd_comment - Required. Please give comment for /etc/passwd.
  -comment          - Optional. Any comment.
  -add              - Optional. Create system role in database.
  -remove           - Optional. Remove system role from database.
  - list_all        - Optional. List all system roles.

For example:
 Creating user role in database:
  [user@host]$ system_role.pl -add -system_role_name app_systems -user_role_name
       unix_role_123 -gid 20000 -group_name app_sys -passwd_comment "Application systems"
       -comment "any comment"
 Remove user role from database:
  [user@host]$ system_role.pl -remove -system_role_name app_systems
 List all user roles in database:
  [user@host]$ system_role.pl - list_all
```

Výpis 13: Spouštěcí skript správy systémových rolí

```
[user@host]$ perl user_map.pl
Usage:
    user_map.pl -add -login_name user123 -user_role_name user_role_123 | -remove -
        login_name user123 -user_role_name user_role_123 | -list_all

  -login_name     - Required. Please give login name.
  -user_role_name - Required. Please give user role name.
  -add            - Optional. Map user to user role.
  -remove         - Optional. Unmap user from user role.
  - list_all      - Optional. List all user maps.

For example:
 Map user to user role:
  [user@host]$ user_map.pl -add -login_name user123 -user_role_name user_role_123
 Unmap user from user role:
  [user@host]$ user_map.pl -remove -login_name user123 -user_role_name user_role_123
```

```
 List  all  user maps:
  [user@host]$ user_map.pl -list_all
```

Výpis 14: Spouštěcí skript mapování uživatelů

```
[user@host]$ perl system_map.pl
Usage:
    system_map.pl -add -fqdn system123 -system_role_name app_systems | -remove -fqdn
        system123 -system_role_name app_systems | -list_all

  -fqdn             - Required. Please give fully  qualified  domain name.
  -system_role_name - Required. Please give system role name.
  -add              - Optional. Map system to system role.
  -remove           - Optional. Unmap system from system role.
  - list_all        - Optional. List  all  system maps.

For example:
 Map system to system role:
  [user@host]$ system_map.pl -add -fqdn system123 -system_role_name app_systems
 Unmap system from system role:
  [user@host]$ system_map.pl -remove -fqdn system123 -system_role_name app_systems
 List  all  system maps:
  [user@host]$ system_map.pl -list_all
```

Výpis 15: Spouštěcí skript mapování systémů

B Systémová dokumentace

B.1 IDM

B.1.1 NAME

IDM - abstract class

B.1.2 SUBROUTINES

new()

Subroutine for creating new constructor.

initialize()

Subroutine for initialization variables.

create_in_repo(@params)

Subroutine for creating new record in database.

delete_from_repo(@id_array)

Delete record from database according to primary keys.

remove($name)

Remove record from database. Checking parameters, then call delete_from_repo().

get_all_parameters(@id_array)

Subroutine for getting all parameters from one record. Parameter is array of primary keys.

get_id($name)

Subroutine for getting primary keys according to name.

get_name(@id_array)

Subroutine for getting name according to primary keys.

get_all_names()

Subroutine for getting all names from the table.

name_exists($name)

Subroutine for checking if name exists.

map_exists(@id_array)

Subroutine for checking if map exists.

get_columns_from(@id_array)

Subroutine for getting a columns from one record on table.

get_full_table()

Subroutine for getting table with all parameters.

print_array(@array)

Subroutine for printing array in TRACE log level.

B.1.3 DEPENDENCIES

Db

Database module.

B.2 Department

B.2.1 NAME

IDM::Department - module for management departments

B.2.2 SYNOPSIS

department.pl - executing script for module IDM::Department

B.2.3 SUBROUTINES

add_new($department_name, $private_key_dir, $department_comment)

Subroutine for adding new department into the database.

department_name
: department name

private_key_dir
: root directory with private keys for users

department_comment
: any comment

B.2.4 DEPENDENCIES

IDM

parent abstract class

B.3 User

B.3.1 NAME

IDM::User - module for work with users

B.3.2 SYNOPSIS

user.pl - executing script for module IDM::User

B.3.3 SUBROUTINES

add_new($department_name, $login_name, $first_name, $last_name, $password_hash, $phone, $email, $user_private_key, $user_comment)

Subroutine for creating new user into the database.

department_name
: Department name

login_name
: User login name

first_name
: User first name

last_name
: User last name

password_hash
: User password hash - unused attribute, maybe used in future for creating accounts without password

phone
: User mobile phone number - unused attribute for future sending messages (password, etc.)

email
: User email address - unused attribute for using in future - sending messages to email address about creating account

user_private_key
: User private RSA/DSA key - unused attribute for using in future. For authorization to systems without password.

user_comment
: Any comment

get_systems($uid)

Subroutine return array of asociated systems with user.

change_password($login_name, $new_password)

Subroutine for changing password on all asociated systems.

B.3.4 DEPENDENCIES

IDM

parent abstract class

B.4 Customer

B.4.1 NAME

IDM::Customer - module for management customers

B.4.2 SYNOPSIS

customer.pl - executing script for module IDM::Customer

B.4.3 SUBROUTINES

add_new($customer_name, $patrol_username, $private_personal_account_flag, $customer_comment)

Subroutine for adding new customer into the database.

customer_name

customer name

patrol_username

patrol username

private_personal_account_flag

flag for private personal accounts for customer - for future using

customer_comment

any comment

B.4.4 DEPENDENCIES

IDM

parent abstract class

B.5 System

B.5.1 NAME

IDM::System - module for work with systems

B.5.2 SYNOPSIS

system.pl - executing script for module IDM::System

B.5.3 SUBROUTINES

add_new($customer_name, $fqdn, $platform, $dhome, $shell, $protocol, $port, $account_type, $account_name, $database_version, $last_jumppoint_name, $jumppoint_flag, $system_comment)

Subroutine for creating new system into the database.

customer_name
: Customer name.

fqdn
: Fully qualified domain name of a system.

platform
: Type of a UNIX platform.

dhome
: Root directory for users home directories.

shell
: Shell - command line interpreter

protocol
: Network protocol for data communication. You can choose from SSH or telnet.

port
: TCP/IP port of a network protocol

account_type
: Type of user account for login to server. You can choose from personal account, system account, root account to root or through sudo command.

account_name
: Name of a system account which will be searched in password database.

database_version
: Password database version of system accounts for authentiction

last_jumppoint_name
: Fully qualified domain name of a last jumppoint system. Jumppoint is previous system from which we connecting to our system.

jumppoint_flag
: Flag if our system is also jumppoint system.

system_comment
: Any comment.

initialize(@args)

Subroutine for initialization and construction supporting modules, e.g.

verify_parameters($customer_id, $fqdn, $platform, $dhome, $shell, $protocol, $port, $account_type, $account_name, $database_version, $last_jumppoint, $jumppoint_flag, $system_comment)

Subroutine for verifying all parameters before adding into the database. A part of is test connection for checking passwords, protocol, account type. After login is also checked root directory of a home directory for users, shell and platform for executing right commands. Default called by add_new().

parse_user($user_db, $uid)

Subroutine for parsing user parameters into the local user database from relational database according to ID. Default called by parsing_parameters().

parse_system($system_db, $system_id)

Subroutine for parsing system parameters into the local system database from relational database according to ID. Default called by parsing_parameters().

parse_system_role($system_role_db, $system_role_id)

Subroutine for parsing system role parameters into the local system role database from relational database according to ID. Default called by parsing_parameters().

parsing_parameters($task, $user_db, $system_db, $system_role_db)

Subroutine for parsing parameters into the local databases. Default called by manage_accounts_on_hosts().

add_user($task, $exp, $uid, $system_id, $system_role_id, $user_db, $system_db, $system_role_db)

Subroutine for adding new user to the system. Default called by check_operation().

delete_user($task, $exp, $uid, $system_id, $system_role_id, $user_db, $system_db, $system_role_db)

Subroutine for deleting user from system. Default called by check_operation().

add_group($task, $exp, $system_id, $system_role_id, $system_db, $system_role_db)

Subroutine for adding new group to the system. Default called by check_operation().

delete_group($task, $exp, $system_id, $system_role_id, $system_db, $system_role_db)

Subroutine for deleting group from system. Default called by check_operation().

password_change($task, $exp, $uid, $system_id, $user_db, $system_db, $new_password)

Subroutine for changing password for user on system. Default called by check_operation().

check_operation($local_db, $task, $exp, $actually_system_id, $user_db, $system_db, $system_role_db, $new_password)

Subroutine for checking operations on system. We can choose from groupadd, useradd, groupdel, userdel and passwd_change. Default called by manage_accounts_on_hosts().

manage_accounts_on_hosts($task, $new_password)

Subroutine for managing all tasks on the systems. It is called from outside perl script when we wan't make changes (all tasks) physicaly on the systems. There is created standalone running process and it will be again forked according to number of a systems. There we call subroutines for connecting to the system, execute commands and disconnect from the system.

verify_home_dir($exp, $dhome, $fqdn)

Subroutine for verifying root directory of a home directory for users. Default called by verify_parameters().

verify_shell($exp, $platform, $shell, $fqdn)

Subroutine for verifying shell command interpreter. Default called by verify_parameters(). ser

verify_platform($exp, $platform, $fqdn)

Subroutine for verifying UNIX platform. Default called by verify_parameters().

connect_expect($exp, $system_db, $system_id, $login_name, $login_password, $root_password)

Subroutine for connecting to the system. Default called by check_account_type() after checking all authentication parameters.

check_account_type($exp, $system_db, $system_id)

Subroutine for checking authentication type and get all credentials. Default called by connect_to_system().

check_password_database($database_version, $fqdn, $account_name)

Subroutine for checking password databases. Default called by check_account_type().

connect_to_system($exp, $system_db, $system_id)

Subroutine for connecting to systems. Default called by manage_accounts_on_hosts().

sub get_cmds()

List of all used commands for all platforms.

Next variables are used by more subroutines, there is description of all variables:

$user_db

Local user database with all users and their atributes

$system_db

Local system database with all systems and their atributes

$system_role_db

Local system role database with all roles and their atributes

B.5.4 DEPENDENCIES

IDM

parent abstract class

Password_db::Fin

Fin user password database

Password_db::Swe

Swe user password database

Expect_common

Module for communication to systems with Expect tool

Task

Module for managing tasks

Parallel::ForkManager

Module for parallel process communication

B.6 User role

B.6.1 NAME

IDM::User_role - module for management user roles

B.6.2 SYNOPSIS

user_role.pl - executing script for module IDM::User_role

B.6.3 SUBROUTINES

add_new($user_role_name, $user_role_comment)

Subroutine for adding new user role into the database.

user_role_name
: user role name

user_role_comment
: any comment

B.6.4 DEPENDENCIES

IDM

parent abstract class

B.7 System role

B.7.1 NAME

IDM::System_role - module for management system roles

B.7.2 SYNOPSIS

system_role.pl - executing script for module IDM::System_role

B.7.3 SUBROUTINES

add_new($user_role_name, $system_role_name, $gid, $group_name, $passwd_comment, $system_role_comment)

Subroutine for adding new system role into the database.

user_role_name
: user role name from user role

system_role_name
: system role name for new role

gid
: identification number of created group on the systems

group_name
: name of created group on the systems

passwd_comment
: group comment in /etc/passwd file for users

system_role_comment
: any comment

B.7.4 DEPENDENCIES

IDM

parent abstract class

B.8 User role mapping

B.8.1 NAME

IDM::User_role_mapping - module for mapping users to user roles

B.8.2 SYNOPSIS

user_map.pl - executing script for module IDM::User_role_mapping

B.8.3 SUBROUTINES

add_new($login_name, $user_role_name)

Subroutine for mapping user into the user role. There is also created new task for creating user to asociated systems.

login_name

User login name

user_role_name

User role name

remove($login_name, $user_role_name)

Subroutine for deleting mapped record. There is also created new task for removing user from asociated systems. There are same atributes as in subroutine add_new().

get_systems($user_role_id)

Subroutine return array of systems mapped according to user_role_id throw system roles.

get_user($user_role_id)

Subroutine returns uid according to user_role_id.

get_user_role($uid)

Subroutine returns user_role_id according to uid.

B.8.4 DEPENDENCIES

IDM

parent abstract class

B.9 System role mapping

B.9.1 NAME

IDM::System_role_mapping - module for mapping systems to system roles

B.9.2 SYNOPSIS

system_map.pl - executing script for module IDM::System_role_mapping

B.9.3 SUBROUTINES

add_new($fqdn, $system_role_name)

Subroutine for mapping system into the system role. There is also created new task for creating asociated users and group on mapped system.

fqdn

Fully qualified domain name of system

system_role_name

System role name

remove($fqdn, $system_role_name)

Subroutine for deleting mapped record. There is also created new task for removing asociated users and group on unmapped system. There are same atributes as in subroutine add_new().

get_users($system_role_id)

Subroutine return array of users mapped according to user_role_id.

get_system($system_role_id)

Subroutine returns system_id according to system_role_id.

get_system_role($system_id)

Subroutine returns system_role_id according to system_id.

B.9.4 DEPENDENCIES

IDM

parent abstract class

B.10 Db

B.10.1 NAME

Db - module for work with relation database SQLite

B.10.2 SUBROUTINES

new()

Constructor for creating new instance of a class.

initialize($db_file)

Subroutine for initialization database.

get_handler()

Subroutine returns DB handle.

B.10.3 DEPENDENCIES

DBI::SQLite

Self Contained RDBMS in a DBI Driver

B.11 Expect common

B.11.1 NAME

Expect_common - module for using Expect utility

B.11.2 SUBROUTINES

new()

Constructor for creating new instance of a class.

spawn()

Subroutine for creating new pseudoterminal and execute program in another process.

connect($cmd, $password)

Subroutine for connecting to the system.

disconnect()

Subroutine for disconnecting from the system.

command_exec($cmd, $values)

Subroutine for executing command.

password_change($cmd, $current_password, $new_password, $error_message)

Subroutine for changing user's password

interact_mode()

Subroutine for interact mode

B.11.3 DEPENDENCIES

Expect

Expect for Perl.

IO::Pty

Pseudo TTY object class.

Data::Dumper

Stringified perl data structures, suitable for both printing and eval.

B.12 Password db

B.12.1 NAME

Password_db - abstract class for getting credentials from user databases

B.12.2 SUBROUTINES

new()

Subroutine for creating new constructor.

B.13 Fin

B.13.1 NAME

Password_db::Fin - modules for getting credentials from Fin database.

B.13.2 SUBROUTINES

query_password($fqdn, $system_user_name)

Subroutine for getting credentials. Returns password.

B.13.3 DEPENDENCIES

Password_db

parent abstract class.

WWW::Mechanize

Handy web browsing in a Perl object.

HTTP::Cookies

HTTP cookie jars.

HTML::Entities

Encode or decode strings with HTML entities.

B.14 Swe

B.14.1 NAME

Password_db::Swe - modules for getting credentials from the Swe database.

B.14.2 SUBROUTINES

query_password($fqdn, $system_user_name)

Subroutine for getting the credentials. Returns password.

B.14.3 DEPENDENCIES

Password_db

parent abstract class.

WWW::Mechanize

Handy web browsing in a Perl object.

WWW::Mechanize::Frames

Perl extension for WWW:Mechanize allowing automatic frames download.

HTTP::Cookies

HTTP cookie jars.

HTML::Entities

Encode or decode strings with HTML entities.

B.15 Security

B.15.1 NAME

Security - module for encryption/decryption text

B.15.2 SYNOPSIS

security.pl - executing script for module Security

B.15.3 SUBROUTINES

new()

Subroutine for creating new constructor.

encrypt($plain_text)

Subroutine for encryption plain text. Returns encrypted text.

decrypt($encrypted_text)

Subroutine for decryption encrypted text. Returns plain text.

B.15.4 DEPENDENCIES

Crypt::CBC

Encrypt Data with Cipher Block Chaining Mode.

B.16 Task

B.16.1 NAME

Task - class for working with the tasks

B.16.2 SUBROUTINES

new()

Subroutine for creating new constructor.

initialize()

Subroutine for initialization variables.

add_new($operation, $system_id, $user_id, $system_role_id)

Add new record into the database. Checking parameters, then call create_in_repo().

remove($operation, $system_id, $user_id, $system_role_id)

Remove record from the database. Checking parameters, then call delete_from_repo().

create_in_repo($operation, $system_id, $user_id, $system_role_id)

Subroutine for creating new record in database.

delete_from_repo($operation, $system_id, $user_id, $system_role_id)

Delete record from database according to keys.

get_all_tasks(@id_array)

Subroutine for getting full table with all tasks.

C Databázové schéma

```
-- --------------------------------------------------------
--
-- Table structure for table 'DEPARTMENT'
--

CREATE TABLE "department" (
    "department_id" INTEGER NOT NULL,
    "department_name" VARCHAR(255) NOT NULL UNIQUE,
    " private_key_dir " VARCHAR(255) NOT NULL,
    "department_comment" VARCHAR(5000),
    PRIMARY KEY ("department_id")
);


-- --------------------------------------------------------
--
-- Table structure for table 'USER'
--

CREATE TABLE "user" (
    "uid" INTEGER NOT NULL,
    "department_id" INTEGER NOT NULL,
    "login_name" VARCHAR(255) NOT NULL UNIQUE,
    "first_name" VARCHAR(255) NOT NULL,
    "last_name" VARCHAR(255) NOT NULL,
    "password_hash" VARCHAR(255),
    "phone" VARCHAR(255),
    "email" VARCHAR(255),
    "user_private_key" VARCHAR(255),
    "user_comment" VARCHAR(5000),
    PRIMARY KEY ("uid"),
    FOREIGN KEY ("department_id") REFERENCES "department" ("department_id")
);


-- --------------------------------------------------------
--
-- Table structure for table 'CUSTOMER'
--

CREATE TABLE "customer" (
    "customer_id" INTEGER NOT NULL,
    "customer_name" VARCHAR(255) NOT NULL UNIQUE,
    "patrol_username" VARCHAR(255),
    "private_personal_account_flag" INTEGER NOT NULL,
    "customer_comment" VARCHAR(5000),
    PRIMARY KEY ("customer_id")
);
```

```
-- ----------------------------------------------------------
--
-- Table structure for table 'SYSTEM'
--

CREATE TABLE "system" (
    "system_id" INTEGER NOT NULL,
    "customer_id" INTEGER NOT NULL,
    "fqdn" VARCHAR(255) NOT NULL UNIQUE,
    "platform" VARCHAR(255) NOT NULL,
    "dhome" VARCHAR(255) NOT NULL,
    "shell" VARCHAR(255) NOT NULL,
    "protocol" INTEGER NOT NULL,
    "port" INTEGER NOT NULL,
    "account_type" INTEGER NOT NULL,
    "account_name" VARCHAR(255),
    "database_version" INTEGER,
    "last_jumppoint" INTEGER,
    "jumppoint_flag" INTEGER NOT NULL,
    "system_comment" VARCHAR(5000),
    PRIMARY KEY ("system_id"),
    FOREIGN KEY ("last_jumppoint") REFERENCES "system" ("system_id"),
    FOREIGN KEY ("customer_id") REFERENCES "customer" ("customer_id")
);

INSERT INTO "system" VALUES( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 );


-- ----------------------------------------------------------
--
-- Table structure for table 'USER_ROLE'
--

CREATE TABLE "user_role" (
    "user_role_id" INTEGER NOT NULL,
    "user_role_name" VARCHAR(255) NOT NULL UNIQUE,
    "user_role_comment" VARCHAR(5000),
    PRIMARY KEY ("user_role_id")
);


-- ----------------------------------------------------------
--
-- Table structure for table 'SYSTEM_ROLE'
--

CREATE TABLE "system_role" (
    "system_role_id" INTEGER NOT NULL,
    "user_role_id" INTEGER NOT NULL,
    "system_role_name" VARCHAR(255) NOT NULL UNIQUE,
    "gid" INTEGER NOT NULL,
    "group_name" VARCHAR(255) NOT NULL,
    "passwd_comment" VARCHAR(255) NOT NULL,
    "system_role_comment" VARCHAR(5000),
```

```
    PRIMARY KEY ("system_role_id"),
    FOREIGN KEY ("user_role_id") REFERENCES "user_role" ("user_role_id")
);


-- ----------------------------------------------------------
--
-- Table structure for table 'USER_ROLE_MAPPING'
--


CREATE TABLE "user_role_mapping" (
    "uid" INTEGER NOT NULL,
    "user_role_id " INTEGER NOT NULL,
    PRIMARY KEY ("uid", "user_role_id"),
    FOREIGN KEY ("uid") REFERENCES "user" ("uid"),
    FOREIGN KEY ("user_role_id") REFERENCES "user_role" ("user_role_id")
);


-- ----------------------------------------------------------
--
-- Table structure for table 'SYSTEM_ROLE_MAPPING'
--


CREATE TABLE "system_role_mapping" (
    "system_id" INTEGER NOT NULL,
    "system_role_id" INTEGER NOT NULL,
    PRIMARY KEY ("system_id", "system_role_id"),
    FOREIGN KEY ("system_id") REFERENCES "system" ("system_id"),
    FOREIGN KEY ("system_role_id") REFERENCES "system_role" ("system_role_id")
);


-- ----------------------------------------------------------
--
-- Table structure for table 'TASK'
--


CREATE TABLE "task" (
    "operation" VARCHAR NOT NULL,
    "system_id" INTEGER NOT NULL,
    "uid" INTEGER,
    "system_role_id" INTEGER,
    PRIMARY KEY ("operation", "system_id", "uid", "system_role_id"),
    FOREIGN KEY ("system_id") REFERENCES "system" ("system_id")
-- FOREIGN KEY ("uid") REFERENCES "user" ("uid"),
-- FOREIGN KEY ("system_role_id") REFERENCES "system_role" ("system_role_id")
);
```

Výpis 16: Databázové schéma datového modelu